Всеволод Несвижский

ПРОГРАММИРОВАНИЕ АППАРАТНЫХ СРЕДСТВ В WINDOWS



Санкт-Петербург
«БХВ-Петербург»
2004

Несвижский Всеволод

Программирование аппаратных средств в Windows. — СПб.: БХВ-Петербург, 2004. — 880 с.: ил.

ISBN 5-94157-476-2

Рассмотрено программирование аппаратных ресурсов в Windows посредством функций BIOS, портов ввода-вывода и программного интерфейса Win32 API. Описаны методы доступа и управления всеми основными устройствами современного персонального компьютера: мышью, клавиатурой, видеоадаптером, звуковой платой, дисковой подсистемой, процессором, шиной, портами и др. Уделено внимание общим методам программирования в Windows, а также различным трюкам и хитростям при написании программ: работе с файлами, взаимодействию в сети, самоликвидации исполняемых файлов, получению данных о USB-устройствах и др. Приведено большое количество простых и понятных примеров, написанных на языках C++ и Assembler.

Для программистов

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Нина Седых*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*



Содержание

Введение

В современном мире, где как грибы после дождя появляются все новые и новые языки программирования, многие люди наивно причисляют себя к программистам, владея одним или двумя популярными скриптовыми имитаторами. К ним я отношу, в первую очередь, такие, как Visual Basic, C# и им подобные. Ни в коем случае не принижая высокий уровень интеграции и удобства данных языков, при всем желании не могу их поставить в один ряд с C, C++ и, тем более, с Assembler. И если в первом случае простота и легкость написания кода приводят к стандартным программам (клонам) и раздутым дистрибутивам, то во втором все зависит от фантазии и мастерства программиста. Конечно, мне могут возразить, что сегодня решающим фактором является время, а не дисковое пространство, но и это составляет всего лишь часть всей правды. Гораздо более важным моментом следует считать надежность и переносимость программного обеспечения. Хорошо конечно, что есть такая операционная система, как Windows, под которую писать программы легко и удобно. А если ее не станет или она полностью изменит свою структуру, а фирма Microsoft в очередной раз "порекомендует" изучить новый скриптовый язык? Разработчикам ничего не останется, как последовать рекомендациям или же, наконец, перейти на полноценный язык программирования. Одним словом, можно сколько угодно гадать о будущем, но оно от этого ничуть не изменится, поэтому изучайте C или C++, не думая о завтрашнем дне.

Книга, которую вы держите в руках, ориентирована на тех, кто любит и ценит C++ и Assembler, кто, несмотря на все заманчивые "предложения" ведущих поставщиков программных средств разработки, выбирает гибкость, мощь и безграничность полета фантазии. Весь представленный материал логически разделен на две части: программирование аппаратного обеспечения и общее программирование в операционных системах Windows. При этом учитываются и самые новые версии Windows, и несправедливо устаревшие.

Часть I полностью посвящена программированию базовых компонентов любого персонального компьютера: мыши, клавиатуры, процессора, системных устройств, дисковой подсистемы, мониторинга питания и температур, видео и звука. Рассматриваются базовые методы программирования данных устройств — посредством набора функций BIOS и прямого доступа через порты ввода-вывода. Уделено должное внимание и программному интерфейсу Win32. Все примеры представлены как на языке Assembler, так и на C++, где это возможно.

Часть II книги содержит очень компактный материал по основным вопросам программирования в Windows. Рассматриваются наиболее важные и востребованные темы: элементы управления, системная оболочка, работа с файлами, Интернет, почта, сеть. Весь этот материал рассчитан на достаточно опытных программистов, имеющих представление о разработке программ под Windows. Практически все примеры написаны на чистом Win32 API, и только некоторые в силу больших размеров используют классы из библиотеки MFC. В последней главе собрано несколько интересных возможностей, логически не вошедших в другие разделы книги и, тем не менее, имеющие большой практический интерес.

Программные требования

Для написания и отладки примеров были использованы оболочки Visual C++ 6.0 и Borland Turbo Assembler 5.0 (TASM). Программисты, работающие в Visual C++ .NET, также могут без проблем выполнять представленные в книге примеры.

Тестирование проводилось в операционных системах Windows ME и Windows 2000. Возникающие проблемы из-за различий в версиях учтены и выделены при рассмотрении материала.

Поддержка

В книге приведено большое количество исходных кодов, размеры которых иногда превышают разумно допустимые для ручного ввода, поэтому читатели могут скопировать их с прилагаемого к книге диска или скачать прямо из Интернета по указанным ниже адресам. Кроме того, для программирования оборудования рекомендуется использовать драйверы, разработанные автором специально для читателей книги. Хочу подчеркнуть, что драйверы представлены исключительно для выполнения примеров из книги и не должны применяться как-либо иначе. Возникающие вопросы и замечания направляйте, пожалуйста, на e-mail: **komarovka@rambler.ru**. Скачать примеры и обновленные драйверы можно по следующим адресам: **http://aspi32.narod.ru** и **http://hardopen.narod.ru**.

ЧАСТЬ I

Работа с аппаратными ресурсами в Windows

ГЛАВА 1

Общие сведения

Прежде чем начинать программирование устройств в операционных системах семейства Windows, необходимо разобраться в основных принципах доступа к аппаратной части компьютера под этими системами. А они, к большому сожалению, довольно скудны и однообразны. Существуют как минимум четыре официальных способа прямого доступа к оборудованию.

1. Первый заключается в обычном использовании набора функции ввода-вывода: `_outp`, `_outpw`, `_outpd`, `_inp`, `_inpw`, `_inpd`. Они входят в состав библиотеки времени выполнения, но их применение очень сильно зависит от операционной системы. Практически все современные системы Windows не позволяют работать с этими функциями в свободном режиме.

2. Второй способ базируется на применении универсальной функции ввода-вывода `DeviceIoControl`. Основное преимущество при ее использовании заключается в однозначной поддержке данной функции всеми системами Windows, начиная с Windows 95 и заканчивая одной из последних — Windows SR3. Но у этой функции есть и серьезный недостаток — очень ограниченный диапазон применения. Несмотря на то, что она позволяет работать с дисковой системой и с современными интерфейсами передачи данных, получить прямой доступ к устройствам с ее помощью не удастся.

3. Третий способ заключается в банальном создании драйвера (например, виртуального драйвера устройства), что позволяет получить неограниченный доступ ко всем устройствам в системе. Кроме того, данный вариант прекрасно будет работать во всех операционных системах Windows. Основной недостаток этого метода заключается в относительной сложности написания самого драйвера, а также необходимость создания отдельного варианта драйвера для каждой операционной системы. Например,

если программа написана под Windows 98 и Windows 2000, то придется писать два разных драйвера под каждую систему.

4. Последний способ заключается в использовании встроенного в Visual C++ макроассемблера. Он не позволяет применять прерывания (Windows будет "зависать"), но вполне неплохо работает с аппаратными портами.

Поскольку каждый из перечисленных способов заслуживает внимания, разберем их подробнее. Сразу замечу, что выбор одного из представленных вариантов будет зависеть в первую очередь от версии операционной системы, а уж затем от решаемых программистом задач. И с этим ничего не поделаешь, т. к. фирма Microsoft с каждым годом все меньше и меньше оставляет возможностей прямого доступа к устройствам. С одной стороны, их можно понять, поскольку эти меры повышают общую надежность системы, но, с другой стороны, блокируют развитие конкурентного и часто более качественного программного обеспечения. Как я уже говорил, сначала мы разберем официальные возможности, а затем рассмотрим существование альтернативного варианта.

1.1. Использование функций ввода-вывода

Существуют шесть функций для работы с портами. Три из них используются для вывода, а три — для ввода данных. К функциям чтения данных относятся:

- _inp — позволяет считать один байт из указанного порта;
- _inpw — позволяет прочитать одно слово из указанного порта;
- _inpd — позволяет прочитать двойное слово из указанного порта.

Все эти функции имеют один аргумент, который должен указывать номер порта, из которого будут прочитаны данные. В зависимости от размера получаемых данных нужно применять ту или иную функцию. В листинге 1.1 показано, как можно работать с этими функциями. Максимальное значение адресуемого порта ограничено числом 65 535, которого вполне достаточно для работы со всеми существующими в системе значениями.

Листинг 1.1. Пример работы с функциями чтения данных из порта

```
// подключаем необходимый файл определений
#include <conio.h>
// прочитаем значение базовой памяти в килобайтах
int GetBaseMemory ()
{
   // объявляем переменные для получения младшего и старшего байтов
   BYTE lowBase = 0, highBase = 0;
```

```
   // читаем информацию из CMOS-памяти
   _outp ( 0x70, 0x15); // записываем номер первого регистра
   lowBase = _inp ( 0x71); // читаем младший байт
   _outp ( 0x70, 0x16); // записываем номер второго регистра
   highBase = _inp ( 0x71); // читаем старший байт
   // возвращаем размер базовой памяти в килобайтах
   return ( ( highBase << 8) | lowBase);
}
// напишем функцию для управления клавиатурой
void KeyBoard_OnOff ( bool bOff)
{
   BYTE state; // текущее состояние
   if ( bOff) // выключить клавиатуру
   {
    // получаем текущее состояние
    state = _inp ( 0x61);
    // устанавливаем бит 7 в 1
    state |= 0x80;
    // записываем обновленное значение в порт
    _outp ( 0x61, state);
   }
   else // включить клавиатуру
   {
    // получаем текущее состояние
    state = _inp ( 0x61);
    // устанавливаем бит 7 в 0
    state &= 0x7F;
    // записываем обновленное значение в порт
    _outp ( 0x61, state);
   }
}
```

Функции записи в порт имеют два аргумента. Первый позволяет указать номер порта, а второй служит для хранения передаваемых данных. К функциям записи данных относятся:

- _outp — позволяет записать один байт в указанный порт;
- _outpw — позволяет записать слово в указанный порт;
- _outpd — позволяет записать двойное слово в указанный порт.

После выполнения все эти функции возвращают переданное значение. Максимальное значение адресуемого порта также ограничено числом 65 535. В листинге 1.2 представлен пример работы с функциями записи.

Листинг 1.2. Пример работы с функциями записи данных в порт

```
// напишем функцию для программного сброса устройства ATA/ATAPI
bool ResetDrive ()
{
   // первое устройство на втором канале ( обычно CD-ROM)
   _outp ( 0x177, 0x08); // пишем команду сброса 08h
   // проверяем результат выполнения
   for ( int i = 0; i < 5000; i++)
   {
     // проверяем бит 7 BUSY
     if ( ( _inp ( 0x177) & 0x80) == 0x00)
        return true; // команда успешно завершена
   }
   return false; // произошла ошибка
}
// напишем функцию для управления лотком CD-ROM
void Eject ( bool bOpen)
{
   int iTimeWait = 50000;
   // формат пакетной команды для открытия лотка
   WORD Eject[6]= { 0x1B, 0, 2, 0, 0, 0 };
   // формат пакетной команды для закрытия лотка
   WORD Close[6]= { 0x1B, 0, 3, 0, 0, 0 };
   // проверяем готовность устройства
   while ( -- iTimeWait > 0)
   {
      // читаем состояние порта
      if ( (_inp ( 0x177) & 0x80 == 0x00) &&
         ( _inp ( 0x177) & 0x08 == 0x00)) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return;
   }
   // выбираем первое устройство на втором канале
   _outp ( 0x176, 0xA0);
   // перед посылкой пакетной команды следует проверить состояние
   iTimeWait = 50000;
   // ожидаем готовности устройства
   while ( -- iTimeWait > 0)
   {
      // читаем состояние порта
      if ( (_inp ( 0x177) & 0x80 == 0x00) &&
         ( _inp ( 0x177) & 0x08 == 0x00)) break;
```

```
    // закончилось время ожидания
    if ( iTimeWait < 1) return;
  }
  // пишем в порт команду пакетной передачи A0h
  _outp ( 0x177, 0xA0);
  // ожидаем готовности устройства к приему пакетной команды
  iTimeWait = 50000;
  // ожидаем готовности устройства
  while ( -- iTimeWait > 0)
  {
    // читаем состояние порта
    if ( (_inp ( 0x177) & 0x80 == 0x00) &&
       ( _inp ( 0x177) & 0x08 == 0x01)) break;
    // закончилось время ожидания
    if ( iTimeWait < 1) return;
  }
  // пишем в порт пакетную команду
  if ( bOpen) // открыть лоток
  {
   for ( int i = 0; i < 6; i++)
   {
     _outpw ( 0x170, Eject[i]); // 12-байтовая команда
   }
  }
  else // закрыть лоток
  {
   for ( int j = 0; j < 6; j++)
   {
     _outpw ( 0x170, Close[j]); // 12-байтовая команда
   }
  }
  // проверяем результат выполнения команды, если нужно
  iTimeWait = 50000;
  // ожидаем готовности устройства
  while ( -- iTimeWait > 0)
  {
    // читаем состояние порта
    if ( (_inp ( 0x177) & 0x80 == 0x00) &&
       (_inp ( 0x177) & 0x01 == 0x00) &&
       ( _inp ( 0x177) & 0x40 == 0x01)) break;
    // закончилось время ожидания
    if ( iTimeWait < 1) return;
  }
}
```

Как видите, работать с функциями ввода-вывода довольно легко и комфортно. Однако прямое их использование в коде возможно лишь в Windows 95. Для работы с ними в профессиональных системах (например, Windows 2000) предварительно придется писать драйвер и загружать его в память перед выполнением кода программы. Как это делается, я расскажу позднее, а сейчас поговорим о функции `DeviceIoControl`.

1.2. Использование функции *DeviceIoControl*

Вообще, данная функция так или иначе использует системные драйверы для доступа к устройствам. Эти драйверы могут входить в состав операционной системы или поставляться разработчиком программного продукта. Универсальность функции состоит в том, что она работает практически с любым драйвером, который поддерживает операции ввода-вывода.

Функция `DeviceIoControl` имеет восемь аргументов. Первый позволяет указать имя драйвера, через который будет осуществляться управление портами (в нашем случае). Второй аргумент представляет собой идентификатор кода требуемой операции, поскольку стандартный драйвер поддерживает несколько (от одной до сотни) операций и требуется конкретно указать, какая из них нужна ему в данный момент. Третий и четвертый аргументы позволяют указать буфер для передаваемых данных и его размер. Их следует применять для операции записи, иначе установить в `NULL`. Пятый и шестой аргументы служат для получения данных от устройства (указатель на буфер данных и размер буфера). Если они не используются, следует установить оба значения в `NULL`. Седьмой аргумент указывает на количество реально полученных данных. Последний аргумент является указателем на структуру `OVERLAPPED`. Она используется при асинхронном вводе-выводе. Далее рассмотрим примеры работы с данной функцией в Windows.

В листинге 1.3 приведен пример функции, позволяющей читать данные с жесткого диска. Она будет работать только в Windows 95/98/ME.

Листинг 1.3. Чтение сектора диска

```
#include "stdafx.h"
#define VWIN32_DIOC_DOS_DRIVEINFO 6 // код функции драйвера
#define CF_FLAG 1 // флаг переноса
// дополнительные структуры
typedef struct _DIOC_REGISTERS
{
        DWORD reg_EBX;
        DWORD reg_EDX;
        DWORD reg_ECX;
```

```
      DWORD reg_EAX;
      DWORD reg_EDI;
      DWORD reg_ESI;
      DWORD reg_Flags;
   } DIOC_REGISTERS;
#pragma pack ( 1)
typedef struct _DATABLOCK
 {
      DWORD dwStartSector; // номер начального сектора
      WORD wNumSectors; // количество секторов
      DWORD pBuffer; // указатель на буфер данных
 } DATABLOCK;
#pragma pack ()
// пишем функцию чтения секторов с диска
bool ReadSector ( unsigned int uDrive, DWORD dwStartSector,
                  WORD wNumSectors, LPBYTE lpBuffer)
{
   HANDLE hDriver;
   DIOC_REGISTERS reg = { 0 };
   DATABLOCK data = { 0 };
   bool bResult;
   DWORD dwResult = 0;
   // инициализируем драйвер
   hDriver = CreateFile ( "\\\\.\\vwin32", 0, 0, NULL, 0,
                          FILE_FLAG_DELETE_ON_CLOSE, 0);
   // если драйвер недоступен, выходим из функции
   if ( hDriver == INVALID_HANDLE_VALUE) return false;
   // заполняем структуру данных DATABLOCK
   data.dwStartSector = dwStartSector;
   data.wNumSectors = wNumSectors;
   data.pBuffer = ( DWORD) lpBuffer;
   // заполняем управляющую структуру
   reg.reg_EAX = 0x7305; // функция 7305h прерывания 21h
   reg.reg_EBX = ( DWORD) &data;
   reg.reg_ECX = -1;
   reg.reg_EDX = uDrive; // номер логического диска
   // вызываем функцию DeviceIoControl
   bResult = DeviceIoControl ( hDriver, VWIN32_DIOC_DOS_DRIVEINFO,
        &reg, sizeof ( reg), &reg, sizeof ( reg), &dwResult, 0);
   // если произошла ошибка, выходим из функции
   if ( !bResult || ( reg.reg_Flags & CF_FLAG))
   {
     CloseHandle ( hDriver);
```

```
      return false;
   }
   return true;
}
// пример использования функции ReadSector для чтения 2-х секторов диска
// номер логического диска может быть следующим: 0 — по умолчанию, 1 — A,
// 2 — B, 3 — C, 4 — D, 4 — E и т. д.
// выделяем память для двух секторов жесткого диска
char* buffer = NULL;
buffer = new char[512*2];
// вызываем функцию чтения секторов
ReadSector ( 3, 0, 2, ( LPBYTE) buffer);
// освобождаем память
delete [] buffer;
```

В профессиональных системах (NT, XP или 2000) использовать `DeviceIoControl` не нужно. Там достаточно открыть функцией `CreateFile` логический диск (даже CD-ROM) и с помощью функции `ReadFile` прочитать данные с диска.

Рассмотрим еще один пример для записи данных на жесткий логический диск (листинг 1.4).

Листинг 1.4. Запись сектора диска

```
// пишем функцию для записи сектора диска
bool WriteSector ( unsigned int uDrive, DWORD dwStartSector,
                   WORD wNumSectors, LPBYTE lpBuffer)
{
HANDLE hDriver;
   DIOC_REGISTERS reg = { 0 };
   DATABLOCK data = { 0 };
   bool bResult;
   DWORD dwResult = 0;
   // инициализируем драйвер
   hDriver = CreateFile ( "\\\\.\\vwin32", 0, 0, NULL, 0,
                          FILE_FLAG_DELETE_ON_CLOSE, 0);
   // если драйвер недоступен, выходим из функции
   if ( hDriver == INVALID_HANDLE_VALUE) return false;
   // заполняем структуру данных DATABLOCK
   data.dwStartSector = dwStartSector;
   data.wNumSectors = wNumSectors;
   data.pBuffer = ( DWORD) lpBuffer;
```

```
    // заполняем управляющую структуру
    reg.reg_EAX = 0x7305; // функция 7305h прерывания 21h
    reg.reg_EBX = ( DWORD) &data;
    reg.reg_ECX = -1;
    reg.reg_EDX = uDrive; // номер логического диска
    reg.reg_ESI = 0x6001;
    // вызываем функцию DeviceIoControl
    bResult = DeviceIoControl ( hDriver, VWIN32_DIOC_DOS_DRIVEINFO,
         &reg, sizeof ( reg), &reg, sizeof ( reg), &dwResult, 0);
    // если произошла ошибка, выходим из функции
    if ( !bResult || ( reg.reg_Flags & CF_FLAG))
    {
      CloseHandle ( hDriver);
      return false;
    }
    return true;
}
```

В следующих главах книги приводятся дополнительные примеры использования функции `DeviceIoControl`.

1.3. Использование драйвера

Данный способ является наиболее гибким и позволяет получить доступ ко всем устройствам в системе. Единственная сложность возникает с написанием самого драйвера. Поскольку все примеры работы с портами в книге основаны на применении драйверов, специально для читателей книги я написал и отладил два драйвера: виртуальный драйвер устройства VxD (Windows 98/ME) и системный драйвер SYS (Windows NT/2000/XP). О том, где их найти, сказано во введении. Здесь же я подробно объясню, как ими пользоваться. Мы напишем два класса для использования этих драйверов. Первый класс рассчитан на работу в Windows 98/ME и представлен в листингах 1.5 и 1.6.

Листинг 1.5. Файл IO32.h

```
// IO32.h: interface for the CIO32 class.
#include <winioctl.h>
// определяем коды функций для чтения и записи
#define IO32_WRITEPORT  CTL_CODE ( FILE_DEVICE_UNKNOWN, 1, \
                                   METHOD_NEITHER, FILE_ANY_ACCESS)
#define IO32_READPORT   CTL_CODE ( ILE_DEVICE_UNKNOWN, 2, \
                                   METHOD_NEITHER, FILE_ANY_ACCESS)
```

```
// объявляем класс
class CIO32
{
public:
       CIO32 ();
       ~CIO32 ();
// общие функции
       bool InitPort (); // инициализация драйвера
       // функция для считывания значения из порта
       bool inPort ( WORD wPort, PDWORD pdwValue, BYTE bSize);
       // функция для записи значения в порт
       bool outPort ( WORD wPort, DWORD dwValue, BYTE bSize);
private:
// закрытая часть класса
       HANDLE hVxD; // дескриптор драйвера
       // управляющая структура
       #pragma pack ( 1)
       struct tagPort32
       {
         USHORT wPort;
         ULONG  dwValue;
         UCHAR  bSize;

       };
       #pragma pack ()
}; // окончание класса
```

Листинг 1.6. Файл IO32.cpp

```
#include "stdafx.h"
#include "IO32.h"
// реализация класса CIO32
// конструктор
CIO32 :: CIO32 ()
{
    hVxD = NULL;
}
// деструктор
CIO32 :: ~CIO32 ()
{
    if ( hVxD) CloseHandle ( hVxD);
    hVxD = NULL;
}
```

```
// функции
bool CIO32 :: InitPort ()
{
    // загружаем драйвер
    hVxD = CreateFile ( "\\\\.\\io32port.vxd", 0, 0, NULL, 0,
                        FILE_FLAG_DELETE_ON_CLOSE, NULL);
    // если драйвер недоступен, прощаемся
    if ( hVxD == INVALID_HANDLE_VALUE)
         return false;
    return true;
}
bool CIO32 :: inPort ( WORD wPort, PDWORD pdwValue, BYTE bSize)
{
    // если драйвер недоступен, прощаемся
    if ( hVxD == NULL) return false;
    DWORD dwReturn;
    tagPort32 port;

    port.bSize = bSize;
    port.wPort = wPort;
    // читаем значение из указанного порта
    return DeviceIoControl ( hVxD, IO32_READPORT, &port,
 sizeof ( tagPort32), pdwValue, sizeof ( DWORD), &dwReturn, NULL);
}
bool CIO32 :: outPort ( WORD wPort, DWORD dwValue, BYTE bSize)
{
    // если драйвер недоступен, прощаемся
    if ( hVxD == NULL) return false;
    DWORD dwReturn;
    tagPort32 port;
    port.bSize   = bSize;
    port.dwValue = dwValue;
    port.wPort   = wPort;
    // записываем значение в указанный порт
    return DeviceIoControl ( hVxD, IO32_WRITEPORT, &port,
                    sizeof ( tagPort32), NULL, 0, &dwReturn, NULL);
}
```

Теперь у нас есть полноценный класс для работы с портами. Перед началом работы нужно вызвать функцию `InitPort` для загрузки виртуального драйвера устройства (файл io32port.vxd). После этого можно писать и читать любые существующие в системе порты ввода-вывода. Каждая из функций `inPort` и `outPort` имеет по три аргумента. Первый позволяет указать номер

порта. Второй аргумент предназначен для передачи или получения значения из порта, а третий определяет размер передаваемых данных. Драйвер поддерживает четыре типа данных: байт (1), слово (2), трехбайтовое значение (3) и двойное слово (4). Не забывайте правильно указывать размер данных, иначе результат будет некорректным. В листинге 1.7 показано, как следует работать с классом CIO32.

Листинг 1.7. Пример использования класса CIO32

```
// объявляем класс
CIO32 io;
// инициализируем драйвер
io.InitPort ();
// теперь можно работать с портами
// для примера включим системный динамик и после 4 секунд выключим
DWORD dwResult = 0;
// читаем состояние порта
io.inPort ( 0x61, &dwResult, 1);
dwResult |= 0x03; // включаем
// записываем значение в порт
io.outPort ( 0x61, dwResult, 1);
// пауза 4 секунды
Sleep ( 4000);
// читаем состояние порта
io.inPort ( 0x61, &dwResult, 1);
dwResult &= 0xFC; // выключаем
// записываем значение в порт
io.outPort ( 0x61, dwResult, 1);
```

Теперь подготовим второй класс CIO32NT, позволяющий работать в профессиональных системах (Windows NT/2000/XP/SR3). В листингах 1.8 и 1.9 представлены файлы определений и реализации класса.

Листинг 1.8. Файл IO32NT.h

```
#include <winioctl.h>
// определяем коды функций драйвера
#define FILE_DEVICE_WINIO 0x00008010
#define WINIO_IOCTL_INDEX 0x810
#define IOCTL_WINIO_ENABLEDIRECTIO   CTL_CODE ( FILE_DEVICE_WINIO, \
              WINIO_IOCTL_INDEX + 2, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_WINIO_DISABLEDIRECTIO  CTL_CODE ( FILE_DEVICE_WINIO, \
              WINIO_IOCTL_INDEX + 3, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

```
// объявляем класс
class CIO32NT
{
public:
       CIO32NT ();
       ~CIO32NT ();
// общие функции
       bool InitPort (); // инициализация драйвера
       // функция для считывания значения из порта
       void inPort ( WORD wPort, PDWORD pdwValue, BYTE bSize);
       // функция для записи значения в порт
       void outPort ( WORD wPort, DWORD dwValue, BYTE bSize);
private:
// закрытая часть класса
       HANDLE hSYS; // дескриптор драйвера
// служебные функции
       // загрузка сервиса
       bool _loadService ( PSTR pszDriver);
       bool _goService (); // запуск сервиса
       bool _stopService (); // остановка сервиса
       bool _freeService (); // закрытие сервиса
}; // окончание класса
```

Листинг 1.9. Файл IO32NT.cpp

```
#include "stdafx.h"
#include "IO32NT.h"
#include <conio.h>
#include <Winsvc.h>
// реализация класса CIO32NT
// конструктор
CIO32NT :: CIO32NT ()
{
    hSYS = NULL;
}
// деструктор
CIO32NT :: ~CIO32NT ()
{
    DWORD dwReturn;
    if ( hSYS != INVALID_HANDLE_VALUE)
    {
     // блокируем драйвер
     DeviceIoControl ( hSYS, IOCTL_WINIO_DISABLEDIRECTIO, NULL,
                             0, NULL, 0, &dwReturn, NULL);
```

```
        CloseHandle ( hSYS); // закрываем драйвер
      }
      // освобождаем системные ресурсы
      _freeService ();
      hSYS = NULL;
}
// функции
bool CIO32NT :: InitPort ()
{
      bool bResult;
      PSTR pszTemp;
      char szExe[MAX_PATH];
      DWORD dwRet;
      // открываем драйвер
      hSYS = CreateFile ( "\\\\.\\IOtrserv", GENERIC_READ | GENERIC_WRITE,
                   0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
      // если не удалось, инициализируем службу сервисов
      if ( hSYS == INVALID_HANDLE_VALUE)
      {
        // получаем имя программы
        if ( !GetModuleFileName ( GetModuleHandle ( NULL), szExe,
                                  sizeof ( szExe)))
              return false;
        // ищем указатель на последнюю косую черту
        pszTemp = strrchr ( szExe, '\\');
        // убираем имя программы
        pszTemp[1] = 0;
        // а вместо него добавляем имя драйвера
        strcat ( szExe, "IOtrserv.sys");
        // загружаем сервис
        bResult = _loadService ( szExe);
        // если ошибка, выходим из функции
        if ( !bResult) return false;
        // запускаем наш сервис
        bResult = _goService ();
        // если ошибка, выходим из функции
        if ( !bResult) return false;
        // открываем драйвер
        hSYS = CreateFile ( "\\\\.\\IOtrserv", GENERIC_READ |
      GENERIC_WRITE, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
         // если не удалось, выходим из функции
         if ( hSYS == INVALID_HANDLE_VALUE) return false;
      }
```

```
    if ( !DeviceIoControl ( hSYS, IOCTL_WINIO_ENABLEDIRECTIO, NULL,
                              0, NULL, 0, &dwRet, NULL))
         return false; // драйвер недоступен
    return true;
}
bool  CIO32NT :: _loadService ( PSTR pszDriver)
{
  SC_HANDLE hSrv;
  SC_HANDLE hMan;
  // на всякий случай выгружаем открытый сервис
  _freeService ();
  // открываем менеджер сервисов
  hMan = OpenSCManager ( NULL, NULL, SC_MANAGER_ALL_ACCESS);
  // создаем объект сервиса из нашего драйвера
  if ( hMan)
  {
   hSrv = CreateService ( hMan, "IOtrserv", "IOtrserv",
   SERVICE_ALL_ACCESS, SERVICE_KERNEL_DRIVER, SERVICE_DEMAND_START,
   SERVICE_ERROR_NORMAL, pszDriver, NULL, NULL, NULL, NULL, NULL);
   // освобождаем менеджер объектов
   CloseServiceHandle ( hMan);
   if ( hSrv == NULL) return false;
  }
  else
   return false;
  CloseServiceHandle ( hMan);
  return true;
}
bool  CIO32NT :: _goService ()
{
   bool bRes;
   SC_HANDLE hSrv;
   SC_HANDLE hMan;
   // открываем менеджер сервисов
   hMan = OpenSCManager ( NULL, NULL, SC_MANAGER_ALL_ACCESS);
   if ( hMan)
   {
     // открываем сервис
     hSrv = OpenService ( hMan, "IOtrserv", SERVICE_ALL_ACCESS);
     // закрываем менеджер сервисов
     CloseServiceHandle ( hMan);
     if ( hSrv)
     {
```

```
      // запускаем сервис
      bRes = StartService ( hSrv, 0, NULL);
      // в случае ошибки закрываем дескриптор
      if( !bRes)
           CloseServiceHandle ( hSrv);
     }
     else
      return false;
   }
   else
    return false;
   return bRes;
}
bool CIO32NT :: _stopService ()
{
   bool bRes;
   SERVICE_STATUS srvStatus;
   SC_HANDLE hMan;
   SC_HANDLE hSrv;
   // открываем менеджер сервисов
   hMan = OpenSCManager ( NULL, NULL, SC_MANAGER_ALL_ACCESS);
   if ( hMan)
   {
     // открываем сервис
     hSrv = OpenService ( hMan, "IOtrserv", SERVICE_ALL_ACCESS);
     // закрываем менеджер сервисов
     CloseServiceHandle ( hMan);
     if ( hSrv)
     {
      // останавливаем сервис
      bRes = ControlService ( hSrv, SERVICE_CONTROL_STOP, &srvStatus);
      // закрываем сервис
      CloseServiceHandle ( hSrv);
     }
     else
      return false;
   }
   else
    return false;
   return bRes;
}
bool CIO32NT :: _freeService ()
{
   bool bRes;
```

```
    SC_HANDLE hSrv;
    SC_HANDLE hMan;
    // останавливаем наш сервис
    _stopService ();
    // открываем менеджер сервисов
    hMan = OpenSCManager ( NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if ( hMan)
    {
      // открываем сервис
      hSrv = OpenService ( hMan, "IOtrserv", SERVICE_ALL_ACCESS);
      // закрываем менеджер сервисов
      CloseServiceHandle ( hMan);
      if ( hSrv)
      {
        // удаляем наш сервис из системы и освобождаем ресурсы
        bRes = DeleteService ( hSrv);
        // закрываем дескриптор нашего сервиса
        CloseServiceHandle ( hSrv);
      }
      else
        return false;
    }
    else
     return false;
    return bRes;
}
// пишем функции ввода-вывода
void CIO32NT :: inPort ( WORD wPort, PDWORD pdwValue, BYTE bSize)
{
    switch ( bSize)
    {
    case 1:
       *pdwValue = _inp( wPort);
        break;
    case 2:
       *pdwValue = _inpw ( wPort);
        break;
    case 4:
        *pdwValue = _inpd ( wPort);
        break;
    }
}
```

```
void CIO32NT :: outPort ( WORD wPort, DWORD dwValue, BYTE bSize)
{
   switch ( bSize)
   {
    case 1:
        _outp ( wPort, ( BYTE) dwValue);
        break;
    case 2:
        _outpw ( wPort, ( WORD) dwValue);
        break;
    case 4:
        _outpd ( wPort, dwValue);
        break;
   }
}
```

Второй класс получился более громоздким за счет особых требований к работе драйверов в профессиональных системах. Пришлось использовать функции менеджера служб SCM (Service Control Manager) для регистрации нашего драйвера в качестве сервиса. Только в этом случае система позволяет получить доступ к аппаратуре. Пример работы с классом CIO32NT приводить не буду, поскольку он ничем не отличается от предыдущего класса.

1.4. Использование ассемблера

Четвертый вариант работы с портами заключается в применении встроенного в Visual C++ ассемблера. Как известно, ассемблер содержит две команды для доступа к портам ввода-вывода: in и out. Однако далеко не в каждой операционной системе удастся воспользоваться этим способом (командами ввода-вывода), поэтому данный вариант рекомендую использовать только в Windows 95/98/ME. Для наглядности рассмотрим пример работы со встроенным ассемблером кода, показанного в листинге 1.10.

Листинг 1.10. Использование встроенного ассемблера в Visual C++

```
// простой пример функции для управления системным динамиком
void PC_dinamik ( bool bOn)
{
   switch ( bOn)
   {
   case true:
     __asm
```

```
    {
      in  al,61h
      or  al, 00000011b // включить динамик
      out 61h, al
    }
    break;
  case false:
    __asm
    {
      in  al, 61h
      and al, 11111100b // отключить динамик
      out 61h, al
    }
    break;
  }
}
```

Встроенный макроассемблер практически ничем не отличается от полноценного ассемблера. Имеются некоторые ограничения, но, в общем, его с успехом можно применить в собственной программе. Сразу хочу заметить, что вызывать прерывания (для упрощения работы) не следует. Это в лучшем случае приведет к фатальной ошибке в программе, а в худшем — "завесит" всю систему.

Вот мы и рассмотрели несколько документированных способов работы с оборудованием в операционных системах Windows. А теперь поговорим о том, есть ли какая-нибудь альтернатива, которая позволяла бы программировать порты ввода-вывода без драйверов и различных ограничений.

1.5. Недокументированный доступ к портам

В многозадачной системе Windows для гарантированно устойчивой (относительно, конечно) работы пришлось использовать так называемый защищенный режим, в котором эффективно разделяются различные выполняемые задачи вместе с используемыми данными. Для выполнения этой задачи потребовалось гораздо больше места под описание адресов, указываемых через сегментные регистры процессора, чем они физически могли предоставить. Было решено в сегментные регистры вместо реальных адресов загружать так называемые селекторы. Селектор представляет собой указатель на 8-байтный блок памяти, который содержит всю необходимую информацию о сегменте. Все эти блоки собраны в таблицы глобальных (GDT — Global Descriptor Table) и локальных (LDT — Local Descriptor Table) дескрипторов. Кроме того, существует и таблица дескрипторов прерываний (IDT — Interrupt Descriptor Table). Селектор состоит из номера дескриптора (адреса)

в таблице (биты 3—15), типа таблицы (1 — LDT, 0 — GDT) (бит 2) и уровня привилегий в битах 0—1 (биты 0—3). Уровень привилегий определяет статус выполняемой задачи. Самая высокая степень привилегий (00b) позволяет программе работать на уровне ядра. Второй уровень (01b) дает полный доступ к аппаратуре. Третий (10b) и четвертый (11b) уровни управляют различными прикладными программами и расширениями. Не буду вдаваться в тонкости, но для доступа к портам ввода-вывода нам нужно попасть на самый высокий уровень (нулевой), для чего необходимо методом перебора получить свободный дескриптор.

Далее мы разберем использование наивысшего уровня привилегий только для операционных систем Windows 95/98/ME. Говорят, есть аналогичный вариант и для профессиональных систем, но мне он неизвестен. Итак, напишем еще один класс для прямого доступа к портам ввода-вывода. В листингах 1.11 и 1.12 представлены соответствующие файлы класса IO32_0, который позволит нам без проблем работать напрямую с любым оборудованием в операционных системах Windows 95/98/ME.

Листинг 1.11. Файл IO32_0.h

```
// объявляем класс
class IO32_0
{
public:
       IO32_0 (); // конструктор
       ~IO32_0 () { } // пустой деструктор
// общие функции
       // прочитать значение из порта
       bool inPort ( WORD wPort, PDWORD pdwValue, BYTE bSize);
       // записать значение в порт
       bool outPort( WORD wPort, DWORD dwValue, BYTE bSize);
private:
#pragma pack ( 1)
// объявляем структуры
// структура для поиска дескриптора в таблице
typedef struct _GDT
{
   WORD Limit; // лимит
   DWORD Base; // база
} GDT;
// описание дескриптора для сегмента данных
typedef struct _GDT_HANDLE
{
   WORD L_0_15; // биты 0-15 лимита
   WORD B_0_15; // биты 0-15 базы сегмента
```

```
   BYTE B_16_23; // биты 16-23 базы сегмента
   BYTE Access : 4; // доступ к сегменту
   BYTE Type : 1; // тип сегмента ( 1 — код, 0 — данные)
   BYTE DPL : 2; // уровень привилегий для дескриптора сегмента
   BYTE IsRead : 1; // проверка наличия сегмента
   BYTE L_16_19 : 4; // биты 16-19 лимита
   BYTE OS : 1; // определяется операционной системой
   BYTE RSV_NULL : 1; // резерв
   BYTE B_32is16 : 1; // разрядность ( 1 — 32-разрядный сегмент, 0 — 16)
   BYTE L_Granul : 1; // гранулярность ( 1 — 4 Кб, 0 - в байтах)
   BYTE B_24_31; // биты 24-31 базы сегмента
} GDT_HANDLE;
// описание дескриптора шлюза
typedef struct _Sluice_Handle
{
   WORD Task_Offset_0_15; // младшее слово смещения для шлюза задачи
   WORD Segment_S; // селектор сегмента
   WORD DWORD_Count : 5; // число двойных слов для работы стека
   WORD NULL_5_7 : 3; // равно 0
   WORD Access : 4; // доступ к сегменту
   WORD Type : 1; // тип шлюза
   WORD DPL : 2; // уровень привилегий для дескриптора шлюза
   WORD IsRead : 1; // проверка наличия сегмента
   WORD Task_Offse_16_31; // старшее слово смещения для шлюза задачи
} Sluice Handle;
#pragma pack ()
   // функция доступа к портам
   bool CallAccess ( PVOID FunctionAddress, WORD wPort, PDWORD pdwValue,
                     BYTE bSize);
}; // окончание класса
```

Листинг 1.12. Файл IO32_0.cpp

```
#include "stdafx.h"
#include "IO32_0.h"
// реализация класса
IO32_0 :: IO32_0 ()
{
   // пустой конструктор
}
// две функции для обработки портов ввода-вывода
__declspec ( naked) void _outPortValue ()
```

```
{
  _asm
  {
    cmp cl, 1 // если размер данных равен байту
    je _Byte // записываем байт
    cmp cl, 2 // если размер данных равен слову
    je _Word // записываем слово
    cmp cl, 4 // если размер данных равен двойному слову
    je _Dword // записываем двойное слово
_Byte:
    mov al, [ebx]
    out dx, al // записываем байт в порт
    retf // выходим
_Word:
    mov ax, [ebx]
    out dx, ax // записываем слово в порт
    retf // выходим

_Dword:
    mov eax, [ebx]
    out dx, eax // записываем двойное слово в порт
    retf // выходим
  }
}
__declspec ( naked) void _inPortValue ()
{
  _asm
  {
    cmp cl, 1 // если размер данных равен байту
    je _Byte // читаем байт
    cmp cl, 2 // если размер данных равен слову
    je _Word // читаем слово
    cmp cl, 4 // если размер данных равен двойному слову
    je _Dword // читаем двойное слово
_Byte:
    in al, dx // читаем байт из порта
    mov [ebx], al
    retf
_Word:
    in ax, dx // читаем слово из порта
    mov [ebx], ax
    retf
_Dword:
    in eax, dx // читаем двойное слово из порта
    mov [ebx], eax
```

```
    retf
  }
}
// функция поиска свободного дескриптора в системе
bool IO32_0 :: CallAccess ( PVOID FunctionAddress, WORD wPort,
                            PDWORD pdwValue, BYTE bSize)
{
  WORD wNum = 1; // счетчик циклов
  WORD Addr[3]; // адрес для нашей задачи
  GDT gdt;
  GDT_HANDLE *pHANDLE;
  // получаем регистр GDTR, начиная с 286 процессора
  _asm sgdt [gdt]
  // переходим ко второму дескриптору сегмента данных
  pHANDLE = ( GDT_HANDLE*) ( gdt.Base + 8);
  // выполняем поиск свободного дескриптора в таблице GDT
  for ( wNum = 1; wNum < ( gdt.Limit / 8); wNum++)
  {
     // если указанные поля структуры равны 0,
     // свободный дескриптор найден
     if ( pHANDLE->IsRead == 0 && pHANDLE->DPL == 0 &&
          pHANDLE->Type == 0 && pHANDLE->Access  == 0)
     {
       // определяем параметры для дескриптора шлюза
       Sluice_Handle *pSluice;
       pSluice = ( Sluice_Handle*) pHANDLE;
       // младшее слово адреса нашей функции
       pSluice->Task_Offset_0_15 = LOWORD ( FunctionAddress);
       // селектор сегмента с наивысшим уровнем привилегий
       pSluice->Segment_S = 0x28;
       pSluice->DWORD_Count = 0;
       pSluice->NULL_5_7 = 0;
       pSluice->Access = 0x0C;
       pSluice->Type = 0;
       pSluice->DPL = 3;
       pSluice->IsRead = 1;
       // старшее слово адреса нашей функции
       pSluice->Task_Offse_16_31 = HIWORD ( FunctionAddress);
       // заполняем адрес для дальнего вызова
       Addr[0] = 0x00;
       Addr[1] = 0x00;
       Addr[2] = ( wNum << 3) | 3;
       // передаем наши параметры
       _asm
```

```
        {
         mov ebx, [pdwValue] // передаваемое значение
         mov cl, [bSize] // размер передаваемого значения
         mov dx, [wPort] // номер порта ввода-вывода
         // выполняем дальний вызов
         call fword ptr [Addr]
        }
        // обнуляем дескриптор
        memset ( pHANDLE, 0, 8);
        return true; // выходим из функции
      }
   // проверяем следующий дескриптор
   pHANDLE++;
   }
   // не найдено ни одного свободного дескриптора
   return false;
}
// общедоступные функции ввода-вывода
bool IO32_0 :: outPort ( WORD wPort, DWORD dwValue, BYTE bSize)
{
   return CallAccess ( ( PVOID) _outPortValue, wPort, &dwValue, bSize);
}
bool IO32_0 :: inPort ( WORD wPort, PDWORD pdwValue, BYTE bSize)
(
   return CallAccess ( ( PVOID) _inPortValue, wPort, pdwValue, bSize);
}
```

Вот у нас и получился полноценный класс для работы с аппаратными портами. Используется он таким же образом, как и предыдущие классы. Принцип работы класса `IO32_0` построен по следующему алгоритму: вначале выполняем поиск свободного дескриптора в таблице GDT. Для получения начального указателя на таблицу применяем системную команду `sgdt`. После этого производим поиск свободного (не используемого системой) дескриптора. Если дескриптор найден, присваиваем ему указатель на заполненный дескриптор шлюза (`Sluice_Handle`). Через структуру шлюза определяем адрес функции, на которую будет передано управление посредством дальнего вызова, а также общие параметры доступа и уровень привилегий. Далее записываем аргументы для функции и непосредственно выполняем дальний вызов, который будет обработан процессором с максимальным приоритетом. Таким образом, мы получим доступ к аппаратным портам, и операционная система не будет мешать этому.

Вот и все, что мне хотелось рассказать о методах работы с портами ввода-вывода в Windows. Вы можете использовать наиболее удобный для вас вариант, чтобы без проблем работать с примерами из последующих глав книги.

ГЛАВА 2

Мышь

Было время, когда мышь была скорее экзотикой, чем необходимым компонентом компьютера. В самом деле, используемые операционные системы не имели графического интерфейса, и все общение с ними было построено исключительно на работе с клавиатурой. Но, как всегда, извечная человеческая лень и стремление приблизить компьютер к реальной жизни подтолкнули разработчиков к созданию графического представления компьютерных данных, отображаемых на дисплее. Операционные системы стали приобретать красивый и удобный интерфейс, в котором вся основная информация предоставлялась пользователю в графическом виде, к слову, более привычном и понятном человеческому глазу. Вот тут-то и появилась необходимость в простых устройствах, позволяющих легко управлять современными компьютерными программами. Одним из таких устройств и явилась мышь. Прошло немного времени, и она завоевала безоговорочную популярность среди обычных пользователей. Естественно, это не могло не отразиться на разработчиках программного обеспечения: им пришлось добавлять в свои программы поддержку мыши. Вот о том, как реализуется работа с мышью в современных операционных системах Windows, я и хотел бы рассказать в данной главе.

Вниманию читателей будет предложено три способа программирования современного устройства мыши:

1. С использованием функций BIOS для прерывания `int 15h`.
2. С использованием аппаратных портов.
3. С использованием возможностей стандартного интерфейса Win32 API.

Каждый из представленных выше способов имеет свои достоинства и недостатки, поэтому выбор оптимального варианта должен отвечать в первую очередь требованиям программиста по использованию мыши в своей программе. Как правило, возможностей Win32 API вполне достаточно. Если же

вы пишете драйвер для мыши или хотите использовать расширенные возможности, недоступные в Win32 API, выбор очевиден — доступ через функции BIOS или аппаратные порты. Использование портов вообще является универсальным и самым мощным способом доступа к мыши (или к любому другому устройству), но для этого необходимо хорошо знать устройство мыши и иметь полную документацию на выбранный стандарт (например, PS/2 или USB). Если по устройству различных мышей информации достаточно, то найти в свободном доступе полную спецификацию для полноценного программирования очень не просто. Такая информация, как правило, предоставляется разработчиками стандартов за деньги.

На сегодняшний день существует несколько основных стандартов для интерфейса мыши: последовательный (Serial Mouse), PS/2 и USB (Universal Serial Bus). Последовательный интерфейс (RS-232C) является одним из самых старых и постепенно вытесняется современными интерфейсами PS/2 и USB. В современном компьютере уже не удастся найти мышь, подключенную к последовательному порту (COM), поскольку ее полностью заменила мышь стандарта PS/2. Использование универсальной последовательной шины USB для подключения мыши является скорее данью моде, а не практической целесообразностью. Этот стандарт не дает важных преимуществ перед PS/2, а только занимает USB-порт. Существует огромное количество других, более важных устройств (принтер, сканер, веб-камера), которые можно подключать к USB-порту. Для мыши специально выделен собственный порт PS/2 и следует использовать именно его.

2.1. Общие сведения

Как уже было отмечено выше, стандарт PS/2 использует последовательный протокол передачи данных. Если посмотреть на заднюю стенку компьютера, то можно увидеть небольшой круглый разъем, имеющий шесть контактных отверстий. Точнее говоря, их там два: для мыши и для клавиатуры, т. к. последняя использует тот же стандарт и протокол для передачи данных. Клавиатура будет рассматриваться в следующей главе, а здесь только замечу, что оба устройства подключаются к контроллеру клавиатуры. Физическая структура PS/2 представлена в табл. 2.1.

Таблица 2.1. *Структура разъема PS/2*

Номер контакта	Описание
1	DATA (линия данных)
2	Не используется
3	GND 0 V (земля)

Таблица 2.1 (окончание)

Номер контакта	Описание
4	+5 V (питание)
5	CLK (синхронизация по времени)
6	Не используется

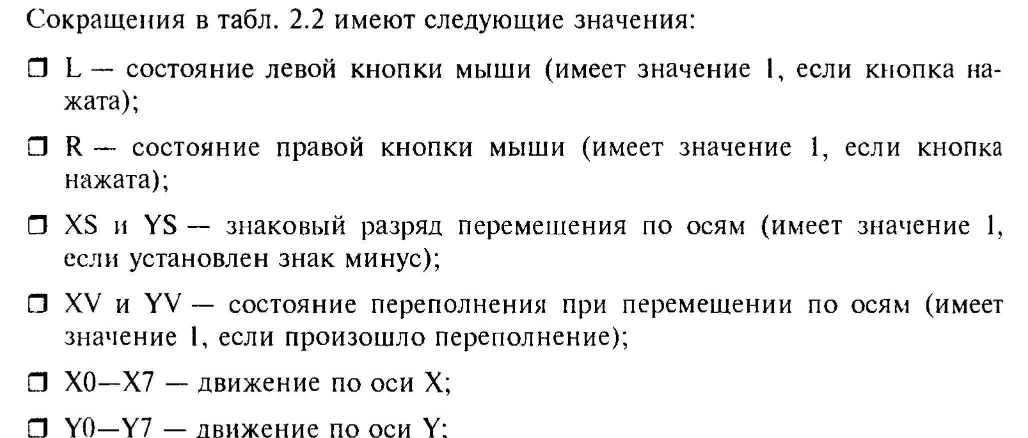
Принцип работы PS/2 мыши достаточно прост: данные со встроенного в мышь микропроцессора поступают на контроллер клавиатуры, находящийся на материнской плате (например, интегрированный контроллер Intel 8042 или Via 8242). На этом этапе происходит также синхронизация последовательно передаваемых данных. Контроллер клавиатуры преобразует поступающую информацию и (посредством BIOS) передает эти данные драйверу мыши. Двунаправленный последовательный обмен данных между мышью и компьютером (контроллером клавиатуры) происходит через линию данных, в зависимости от сигнала синхронизации. Например, установка компьютером нулевого значения на линии синхронизации позволяет запретить передачу данных. Устройство постоянно генерирует сигнал синхронизации. Прежде чем передать данные, компьютер устанавливает линию синхронизации в 0. Далее следует передача данных, линия данных устанавливается в 0, а линия синхронизации — в 1. Данные передаются в следующем порядке:

1. Стартовый бит (всегда должен быть равен 0).
2. Восемь бит данных, начиная с младшего.
3. Один бит четности.
4. Стоповый бит (всегда равен 1).

Любая мышь PS/2 имеет как минимум два счетчика, отслеживающих движение: по оси X и по оси Y. Значение каждого счетчика определяется 9-разрядным двоичным числом (старший разряд является знаковым) и связанным с ним флагом переполнения. Каждое значение счетчика и состояние кнопок мыши передаются на компьютер в виде пакета данных. Размер одного такого пакета данных равен 3 байтам. Формат пакета данных, передаваемых двухкнопочной мышью PS/2, представлен в табл. 2.2. Для мышей с тремя и более кнопок формат пакета данных будет расширен до 4 байтов.

Таблица 2.2. Формат пакета данных мыши PS/2

Бит Байт	7	6	5	4	3	2	1	0
1	YV	XV	YS	XS	1	0 (M)	R	L
2	X7	X6	X5	X4	X3	X2	X1	X0
3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0



Сокращения в табл. 2.2 имеют следующие значения:

- L — состояние левой кнопки мыши (имеет значение 1, если кнопка нажата);
- R — состояние правой кнопки мыши (имеет значение 1, если кнопка нажата);
- XS и YS — знаковый разряд перемещения по осям (имеет значение 1, если установлен знак минус);
- XV и YV — состояние переполнения при перемещении по осям (имеет значение 1, если произошло переполнение);
- X0—X7 — движение по оси X;
- Y0—Y7 — движение по оси Y;
- M — состояние средней кнопки мыши. Для двухкнопочной мыши это значение равно 0.

После посылки очередного пакета счетчики движения сбрасываются в 0, а диапазон возможных значений лежит в промежутке между −255 и 255. Если диапазон превышен, устанавливается бит переполнения и дальнейшая работа счетчиков движения возможна только после сброса. Сброс счетчиков происходит также при получении устройством мыши любой команды от компьютера, кроме `Resend` (`0xFE`). Описание команд приводится далее в *разд. 2.3*.

Мышь PS/2 имеет два важных свойства, определяющих ее работу: частоту дискретизации и чувствительность. Частотой дискретизации называется число выборок в секунду, передаваемых устройством мыши на компьютер. Поддерживаются следующие значения частоты: 200, 100, 80, 60, 40, 20 и 10 выборок за одну секунду. По умолчанию используется 100 выборок в секунду. Изменить это значение можно с помощью команды `Set Sample Rate` (`0xF3`).

Под чувствительностью понимается количество отсчетов, фиксируемых счетчиками движения на расстоянии одного миллиметра. Стандартом оговорены несколько уровней чувствительности: 1, 2, 4 и 8 отсчетов. По умолчанию используются 4 отсчета на один миллиметр. Кроме того, на чувствительность может влиять так называемый режим масштабирования. По умолчанию он равен 1 : 1, не влияя на чувствительность мыши. Можно также установить дополнительный масштаб 1 : 2, который существенно увеличивает чувствительность устройства. В табл. 2.3 представлены значения счетчика в режиме масштабирования 1 : 2. Принцип работы масштабирования устроен на дополнительном алгоритме пересчета, применяемом к счетчикам движения, перед началом передачи данных на компьютер.

Таблица 2.3. *Значения счетчика перемещения в режиме 1 : 2*

Значение счетчика движения	Пересчитанное значение
0	0
1	1
2	1
3	3
4	6
5	9
N > 5	2 × N

Мышь PS/2 поддерживает четыре стандартных режима работы.

1. Режим сброса может быть установлен после подачи питания или с помощью команды Reset (0xFF).
2. Потоковый режим задан по умолчанию и является основным режимом работы мыши PS/2. В нем осуществляется передача данных между устройством мыши и компьютером. Если мышь находится в дистанционном режиме, можно применить команду Set Remote Mode (0xF0) для перевода ее в потоковый режим.
3. Дистанционный режим позволяет управлять передачей данных от мыши на компьютер только по запросу последнего. Компьютер может запросить тип устройства (Get Device ID), состояние (Status Request), а также данные (Read Data). Для ввода мыши в данный режим используется команда Set Remote Mode (0xF0).
4. Эхо-режим позволяет любой посланный компьютером байт вернуть обратно. Может применяться для тестирования устройства мыши. Включить данный режим можно с помощью команды Set Wrap Mode (0xEE). Для выхода из режима следует применить команду Reset Wrap Mode (0xEC) или Reset (0xFF).

После входа в режим сброса устройство мыши выполняет самодиагностику и устанавливает следующие параметры: частота дискретизации равна 100 выборок в секунду, чувствительность — 4 отсчета на один миллиметр, масштабирование — 1 : 1, передача данных отключена. В случае успешного самотестирования мышь посылает компьютеру значение 0xAA, а в случае ошибки — значение 0xFC. Кроме того, после кода тестирования (0xAA или 0xFC) мышь посылает идентификатор устройства, всегда равный 0x00. В любом случае при получении данных от мыши общий контроллер клавиатуры вырабатывает стандартное прерывание IRQ12, которое закреплено за мышью и является постоянным для большинства компьютеров.

Вот и все общие сведения о мыши стандарта PS/2, с которыми мне хотелось познакомить читателя. Детальное описание команд приводится в *разд. 2.3*. Теперь же перейдем к программированию любого типа мыши, используя возможности базовой системы ввода-вывода (BIOS).

2.2. Использование функций BIOS

Для поддержки работы с мышью используется прерывание BIOS `int 15h`. Набор значений, поддерживающий мышь PS/2, лежит в диапазоне от `C200h` до `C209h`. Старший байт определяет номер функции `C2h`, а младший — номер подфункции. В табл. 2.4 представлен список всех подфункций, управляющих работой мыши PS/2.

Таблица 2.4. *Подфункции BIOS для PS/2*

Значение подфункции	Описание
00h	Включение или выключение устройства мыши
01h	Сброс устройства мыши
02h	Установка частоты дискретизации
03h	Установка чувствительности
04h	Получение идентификатора устройства мыши
05h	Инициализация устройства мыши
06h	Установка режима масштабирования и получение информации о состоянии
07h	Указание нового адреса обработчика для устройства мыши
08h	Запись данных в порт устройства мыши
09h	Чтение данных из порта устройства мыши

Последние две подфункции (`08h` и `09h`) являются необязательными и могут не поддерживаться конкретным устройством мыши.

Рассмотрим подробнее каждую подфункцию.

2.2.1. Подфункция *00h*

Данная подфункция предназначена для включения или выключения устройства мыши. Она является обязательной для всех устройств PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C200h.
2. В регистр BH заносится значение операции: 00h (выключить) или 01h (включить).
3. Вызывается прерывание BIOS int 15h.

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5.

Таблица 2.5. *Коды ошибок*

Код ошибки	Описание
00h	Операция успешно завершена
01h	Недопустимый номер функции (подфункции)
02h	Использование недопустимых значений параметров (например, ввод значения 04h в регистр BH)
03h	Ошибка в интерфейсе PS/2
04h	Требуется повторить передачу данных для устройства мыши
05h	Отсутствует необходимый обработчик (драйвер) мыши

Приведу простой пример использования подфункции 00h для запрещения работы мыши (листинг 2.1).

Листинг 2.1. Использование подфункции 00h

```
mov AX, 0C200h; помещаем в регистр AX номер функции
mov BH, 00h   ; заносим нулевое значение для отключения мыши
int 15h       ; вызываем прерывание BIOS
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

2.2.2. Подфункция *01h*

Подфункция 01h предназначена для сброса мыши PS/2. Она является обязательной для всех устройств PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C201h.
2. Вызвать прерывание BIOS int 15h.

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5. Если функция завершена успешно, флаг CF очищен, а регистр BX будет содержать следующее значение: старший байт регистра (BH) — идентификатор устройства (как правило, 00h), младший байт регистра (BL) — возвращаемое значение (для мыши PS/2 равно AAh).

Простой пример использования подфункции 01h для сброса устройства мыши показан в листинге 2.2.

Листинг 2.2. Использование подфункции 01h

```
mov AX, 0C201h; помещаем в регистр AX номер функции
int 15h       ; вызываем прерывание BIOS
jc  ErrorHnd  ; если произошла ошибка, вызываем обработчик
cmp BL, 0AAh  ; проверяем, если нужно, возвращаемое значение
```

2.2.3. Подфункция *02h*

Данная подфункция предназначена для установки частоты дискретизации мыши PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C202h.
2. В регистр BH заносится значение кода для частоты дискретизации. Возможные значения представлены в табл. 2.6.
3. Вызывается прерывание BIOS int 15h.

***Таблица 2.6.** Значения кодов частоты дискретизации*

Значение кода	Частота дискретизации, выборок/сек
00h	10
01h	20
02h	40
03h	60
04h	80
05h	100
06h	200

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5.

Рассмотрим небольшой пример использования функции C202h для установки максимального значения частоты дискретизации мыши PS/2. К слову, эта операция существенно улучшит работу мыши, особенно в графических и чертежных программах. В листинге 2.3 показан пример работы с данной подфункцией.

Листинг 2.3. Использование подфункции 02h

```
mov AX, 0C202h; помещаем в регистр AX номер функции
mov BH, 06h   ; частота дискретизации 200 выборок в секунду
int 15h       ; вызываем прерывание BIOS
jc  ErrorHnd  ; если произошла ошибка, вызываем обработчик
```

2.2.4. Подфункция *03h*

Подфункция предназначена для управления разрешением (чувствительностью) мыши PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C203h.
2. В регистр BH заносится значение кода для устанавливаемого разрешения. Возможные значения представлены в табл. 2.7.
3. Вызывается прерывание BIOS int 15h.

Таблица 2.7. *Значения кодов разрешения*

Значение кода	Разрешение, отсчетов/мм
00h	1
01h	2
02h	4
03h	8

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5.

Простой пример использования подфункции 03h для установки нового разрешения мыши показан в листинге 2.4.

Листинг 2.4. Использование подфункции 03h

```
mov AX, 0C203h; помещаем в регистр AX номер функции
mov BH, 01h  ; разрешение 2 отсчета на один миллиметр
int 15h      ; вызываем прерывание BIOS
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

2.2.5. Подфункция *04h*

Подфункция 04h служит для получения идентификатора устройства, по которому можно определить тип подключенного устройства.

Использование:

1. В регистр AX следует поместить полный код функции, равный C204h.
2. Вызвать прерывание BIOS int 15h.

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5. Если ошибок нет, флаг CF будет очищен, а в старший байт регистра BX будет помещено значение идентификатора.

Пример использования подфункции 04h для получения идентификатора устройства мыши показан в листинге 2.5.

Листинг 2.5. Использование подфункции 04h

```
TypeID db ?  ; переменная для хранения идентификатора мыши
...          ; общий код программы
mov AX, 0C204h; помещаем в регистр AX номер функции
int 15h      ; вызываем прерывание BIOS
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
mov TypeID, BH; помещаем полученное значение идентификатора в переменную
```

2.2.6. Подфункция *05h*

Данная подфункция предназначена для инициализации интерфейса мыши PS/2. После ее выполнения устройство мыши будет иметь следующие параметры: выключено, разрешение — 4 отсчета на миллиметр, частота дискретизации — 100 выборок в секунду, масштабирование — 1 : 1.

Использование:

1. В регистр AX следует поместить полный код функции, равный C205h.
2. В регистр BH заносится размер пакета данных (от 1 до 8 байтов).
3. Вызывается прерывание BIOS int 15h.

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5. Простой пример использования подфункции 05h для инициализации мыши показан в листинге 2.6.

Листинг 2.6. Использование подфункции 05h

```
mov AX, 0C205h; помещаем в регистр AX номер функции
mov BH, 08h   ; размер пакета данных
int 15h       ; вызываем прерывание BIOS
jc  ErrorHnd  ; если произошла ошибка, вызываем обработчик
```

2.2.7. Подфункция *06h*

Подфункция предназначена для передачи устройству мыши PS/2 расширенной команды. Под расширенной командой понимается одна из трех возможных операций: определение состояния устройства, установка масштабирования в 1 : 1, установка масштабирования в 2 : 1.

Использование:

1. В регистр AX следует поместить полный код функции, равный C206h.
2. В регистр BH заносится код расширенной команды. Возможные значения представлены в табл. 2.8.
3. Вызывается прерывание BIOS int 15h.

Таблица 2.8. *Значения кодов расширенной команды*

Код команды	Описание
00h	Возвратить состояние устройства
01h	Установить масштаб 1 : 1
02h	Установить масштаб 2 : 1

Выход:

При использовании кода расширенной команды, равного 01h или 02h, следует проверить флаг CF. В случае возникновения ошибки флаг будет уста-

новлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5. Если же использовалась расширенная команда с кодом 00h, тогда младший байт регистра BX будет содержать битовую маску состояния устройства (табл. 2.9), младший байт регистра CX — текущее разрешение (см. табл. 2.7); а младший байт регистра DX — текущую частоту дискретизации (см. табл. 2.6).

Таблица 2.9. *Битовая маска состояния устройства*

Бит	Описание
0	Статус правой кнопки (1 — нажата, 0 — отпущена)
1	Статус средней кнопки (1 — нажата, 0 — отпущена)
2	Статус левой кнопки (1 — нажата, 0 — отпущена)
3	Зарезервирован
4	Масштаб (1 для 1 : 1, 0 для 2 : 1)
5	Состояние устройства (1 — включено, 0 — выключено)
6	Режим работы (1 — дистанционный, 0 — потоковый)
7	Зарезервирован

Пример использования подфункции 06h для установки масштаба 2 : 1 устройства мыши показан в листинге 2.7.

Листинг 2.7. Использование подфункции 06h

```
mov AX, 0C206h; помещаем в регистр AX номер функции
mov BH, 02h   ; код расширенной команды (масштаб 2:1)
int 15h       ; вызываем прерывание BIOS
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

2.2.8. Подфункция *07h*

Подфункция 07h позволяет указать новый адрес обработчика прерываний для устройства мыши PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C207h.
2. В ES:BH заносится дальний указатель (FAR) на собственный обработчик. Для отмены пользовательского обработчика следует указать значение 0000h:0000h.
3. Вызывается прерывание BIOS int 15h.

Выход:

После выполнения функции следует проверить флаг CF. В случае возникновения ошибки флаг будет установлен в 1, а в регистр AH будет помещен код ошибки. Возможные коды ошибок перечислены в табл. 2.5.

Простой пример использования подфункции 07h для инициализации мыши показан в листинге 2.8.

Листинг 2.8. Использование подфункции 07h

```
mov AX, 0C207h; помещаем в регистр AX номер функции
les BX, MyHand; указываем адрес и смещение нашего обработчика
int 15h       ; вызываем прерывание BIOS
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик ошибок
```

2.2.9. Подфункция *08h*

Данная подфункция предназначена для записи в порт устройства любого значения размером в один байт. Данная подфункция является необязательной и может не поддерживаться стандартными устройствами мыши PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C208h.
2. В регистр BL записывается значение передаваемого байта.
3. Вызывается прерывание BIOS int 15h.

Выход:

Проверка возвращаемого значения не определена.

Хотя эта подфункция и не имеет серьезных шансов на практическое использование, приведу простой пример ее использования в листинге 2.9.

Листинг 2.9. Использование подфункции 08h

```
mov AX, 0C208h; помещаем в регистр AX номер функции
mov BL, 64h  ; записываем в порт значение 100
int 15h      ; вызываем прерывание BIOS
```

2.2.10. Подфункция *09h*

Подфункция предназначена для чтения из порта устройства текущей информации из трех байт о состоянии мыши. Данная подфункция является необязательной и может не поддерживаться стандартными устройствами мыши PS/2.

Использование:

1. В регистр AX следует поместить полный код функции, равный C209h.
2. Вызывается прерывание BIOS int 15h.

Выход:

После завершения подфункции информация будет распределена следующим образом: BL — первый байт, CL — второй байт и DL — третий байт.

Простой пример использования подфункции 07h для инициализации мыши показан в листинге 2.10.

Листинг 2.10. Использование подфункции 09h

```
Stat_1 db ?  ; переменная для хранения первого байта
Stat_2 db ?  ; переменная для хранения второго байта
Stat_3 db ?  ; переменная для хранения третьего байта
...          ; общий код программы
mov AX, 0C209h; помещаем в регистр AX номер функции
int 15h      ; вызываем прерывание BIOS
mov Stat_1, BL; сохраняем полученную информацию
mov Stat_2, CL
mov Stat_3, DL
```

На этом можно завершить рассмотрение средств BIOS по управлению устройством мыши PS/2. Информация довольно скудная, что связано с отсутствием какой-либо полноценной документации по данному вопросу. Для тех, кто пишет программы под DOS, могу посоветовать самостоятельно ознакомиться с набором соответствующих функций (с использованием прерывания int 33h), поддерживающих работу мыши.

2.3. Использование портов

Доступ к устройству посредством аппаратных портов всегда был и останется самым мощным и универсальным. Он дает возможность непосредственно обратиться к контроллеру мыши, интегрированному на материнской плате. Самыми важными его преимуществами являются следующие:

- поддержка любых новых возможностей, появляющихся в расширениях стандарта;
- отсутствие обращений к прерываниям BIOS и DOS;
- заметное повышение скорости в приложениях реального времени.

Наряду с достоинствами, есть и несколько недостатков:

- необходимо отличное знание самого устройства и наличие документации к нему;

- полный учет всех возможных вариантов взаимодействия по требуемому протоколу;
- владение языком Assembler или С (С++). К сожалению, современные языки (Visual Basic, С#, Delphi, Java и т. д.) все дальше и дальше "уводят" начинающих программистов от интересного, но сложного, к более простому, но очень ограниченному.

Мне бы хотелось рассмотреть документированные способы работы с мышью PS/2. Еще раз напомню, что информации по данной теме очень мало, и здесь представлена вся имеющаяся в свободном доступе.

Итак, для непосредственной работы с мышью PS/2 используются два порта контроллера клавиатуры, адресуемые через регистры `64h` и `60h`. Регистр `64h` имеет двойное назначение. В режиме чтения он служит регистром статуса и позволяет получить текущую информацию о состоянии мыши. Этот регистр имеет размер 8 бит и представлен в табл. 2.10.

***Таблица 2.10.** Регистр состояния (`64h`)*

Бит	Описание
0	Наличие данных в выходном буфере мыши (0 — выходной буфер пустой, 1 — в буфере есть данные)
1	Наличие данных во входном буфере мыши (0 — входной буфер пустой, 1 — в буфере есть данные)
2	Результат самотестирования (0 — сброс, 1 — тест прошел успешно)
3	Использование портов контроллером клавиатуры (0 — запись в порт `60h`, 1 — запись в порт `64h`)
4	Состояние клавиатуры (0 — заблокирована, 1 — включена)
5	Наличие данных в выходном буфере мыши (0 — выходной буфер пустой, 1 — в буфере есть данные)
6	Ошибка тайм-аута (0 — ошибка отсутствует, 1 — ошибка). В случае ошибки следует повторить передачу данных, используя команду `Resend`
7	Ошибка четности (0 — ошибка отсутствует, 1 — ошибка) указывает на последнюю ошибку, произошедшую при передаче данных

В режиме записи регистр `64h` служит для передачи команд контроллеру клавиатуры. Через запись в этот регистр команды осуществляется управление контроллером клавиатуры. Назначения битов в командном регистре перечислены в табл. 2.11.

Список команд, используемых для мыши PS/2, перечислен в табл. 2.12.

Таблица 2.11. *Регистр команд (64h)*

Бит	Описание
0	Прерывания для клавиатуры (0 — отключить, 1 — включить)
1	Прерывания для мыши (0 — отключить, 1 — включить)
2	Системный флаг (1 — инициализация через самотестирование, 0 — инициализация по питанию)
3	Не используется
4	Доступ к клавиатуре (0 — открыт, 1 — закрыт)
5	Доступ к мыши (0 — открыт, 1 — закрыт)
6	Трансляция скан-кодов (0 — не использовать, 1 — использовать)
7	Резерв

Таблица 2.12. *Команды управления для мыши*

Код команды	Описание
20h	Прочитать регистр команд
60h	Записать байт в регистр команд
A7h	Отключить порт мыши PS/2
A8h	Включить порт мыши PS/2
A9h	Выполнить самотестирование мыши (байт на выходе равен 00h)
AAh	Выполнить самотестирование контроллера (байт на выходе равен 55h)
C0h	Прочитать входной порт
C1h	Скопировать из входного порта младший разряд байта
C2h	Скопировать из входного порта старший разряд байта
D0h	Прочитать расширенную информацию для выходного порта (0 бит — сброс процессора, 1 бит — канал A20 включен, 2 бит — передача данных от мыши PS/2, 3 бит — синхросигнал от мыши PS/2, 4 бит — выходной буфер заполнен, 5 бит — выходной буфер мыши PS/2 заполнен, 6 бит — синхросигнал для клавиатуры, 7 бит — передача данных от клавиатуры)
D1h	Записать байт данных в выходной порт
D3h	Записать байт данных в выходной буфер мыши PS/2
D4h	Записать байт данных для мыши PS2

Для обмена данными с мышью используется регистр 60h, называемый регистром данных. После выполнения какой-либо команды в данный регистр помещается код ошибки, по которому можно судить о результате операции. Возможные значения кодов ошибок представлены в табл. 2.13. Если команда управления является расширенной (имеет размер два байта), то первый байт команды должен быть записан в командный регистр, а второй байт — в регистр данных.

Таблица 2.13. *Коды ошибок мыши*

Код ошибки	Описание
AAh	Самотестирование контроллера успешно выполнено
FAh	Команда успешно передана
FCh	Ошибка самотестирования мыши
FEh	Запрос на повторную передачу данных (команда Resend)

После того, как были рассмотрены основные команды контроллера клавиатуры для управления устройством мыши PS/2, приведу несколько примеров их использования. Пример кода, позволяющий протестировать интерфейс мыши PS/2, показан в листинге 2.11.

Листинг 2.11. Тестирование интерфейса мыши PS/2

```
@repeat:
in  AL, 64h ; получаем состояние порта
and AL, 10b ; проверяем наличие данных во входном буфере
jnz @repeat ; буфер занят, пытаемся повторить запрос
mov AL, A9h ; запишем команду самотестирования
out 64h, AL ; запишем значение в командный регистр
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
```

Этот же пример для C++ показан в листинге 2.12.

Листинг 2.12. Тестирование интерфейса мыши PS/2 в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере
while ( -- iTimeWait > 0)
```

```
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x02) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду самотестирования
outPort ( 0x64, 0xA9, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x02) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// самотестирование мыши успешно завершено
```

Если вы заметили, мы постоянно проверяем регистр статуса (бит 2) на наличие данных во входном буфере. В реальных программах дополнительно следует проверять ошибки тайм-аута (бит 6) и четности (бит 7).

Рассмотрим еще один пример, позволяющий программно отключить мышь PS/2 (листинг 2.13).

Листинг 2.13. Отключение мыши PS/2

```
@repeat:
in  AL, 64h ; получаем состояние порта
and AL, 10b ; проверяем наличие данных во входном буфере
jnz @repeat ; буфер занят, пытаемся повторить запрос
mov AL, A7h ; запишем команду отключения мыши
out 64h, AL ; запишем значение в командный регистр
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
```

Этот же пример для C++ представлен в листинге 2.14.

Листинг 2.14. Отключение мыши PS/2 в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
```

```
// проверяем наличие данных во входном буфере
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x64, &dwResult, 1);
      if ( (dwResult & 0x02) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду отключения мыши PS/2
outPort ( 0x64, 0xA7, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x64, &dwResult, 1);
      if ( (dwResult & 0x02) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// мышь успешно заблокирована
```

В обоих примерах используются команды управления контроллером клавиатуры, но для мыши PS/2 существует еще дополнительный набор команд, разработанный специально для нее. Некоторые производители мышей могут добавлять свои специфические команды, которые можно найти в технической документации, предоставляемой разработчикам. Список всех стандартных команд для мыши PS/2 приведен в табл. 2.14.

Таблица 2.14. *Набор команд для мыши PS/2*

Код команды	Описание
FFh	Reset
FEh	Resend
F6h	Set Defaults
F5h	Disable
F4h	Enable
F3h	Set Sample Rate
F2h	Read Device Type
F0h	Set Remote Mode

Таблица 2.14 *(окончание)*

Код команды	Описание
EEh	Set Wrap Mode
ECh	Reset Wrap Mode
EBh	Read Data
EAh	Set Stream Mode
E9h	Status Request
E8h	Set Resolution
E7h	Set Scaling 2:1
E6h	Set Scaling 1:1

Процесс вызова каждой дополнительной команды PS/2 немного отличается от прямого управления контроллером клавиатуры. Вначале выполняется команда D4h (записать байт данных для мыши PS2), и только потом в регистр данных записывается код команды мыши PS/2 (например, FFh для сброса мыши). Говоря проще, каждая дополнительная команда PS/2 имеет размер два байта (D4h и любой код команды), где первый байт записывается в регистр команд (64h), а второй — в регистр данных (60h). Если команда PS/2 требует передачи дополнительных данных (например, нового значения частоты дискретизации), то эта информация должна быть записана в регистр данных (60h), а первый байт D4h также записывается в командный регистр. Далее будут приведены примеры использования команд PS/2, где можно будет наглядно убедиться в этом. А теперь рассмотрим набор команд управления мышью PS/2 более подробно.

2.3.1. Команда *Reset* (*FFh*)

Данная команда предназначена для программного сброса устройства мыши PS/2. После успешного выполнения команды (через 300—500 мс) мышь возвращает код FAh и комбинацию AAh и 00h. После этого мышь устанавливает режим сброса. В листинге 2.15 приводится упрощенный пример кода для программного сброса мыши PS/2.

Листинг 2.15. Выполнение сброса контроллера мыши PS/2

```
@repeat:
in  AL, 64h ; получаем состояние порта
and AL, 10b ; проверяем наличие данных во входном буфере
jnz @repeat ; буфер занят, пытаемся повторить запрос
mov AL, 0D4h; запишем команду передачи байта
```

```
out 64h, AL ; запишем значение в командный регистр
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
mov AL, FFh ; запишем код команды Reset
out 60h, AL ; в регистр данных
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
```

Дополнительно, после выполнения команды сброса можно проверить регистр `60h` на наличие кода `FAh`. Аналогичный пример для C++ показан в листинге 2.16.

Листинг 2.16. Выполнение сброса контроллера мыши PS/2 в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x64, &dwResult, 1);
      if ( (dwResult & 0x02) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду передачи данных
outPort ( 0x64, 0xD4, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x64, &dwResult, 1);
      if ( (dwResult & 0x02) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду Reset
outPort ( 0x60, 0xFF, 1);
iTimeWait = 50000;
```

```
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x64, &dwResult, 1);
      if ( (dwResult & 0x02) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// программный сброс мыши завершен
```

2.3.2. Команда *Resend* (*FEh*)

Команда предназначена для проверки данных, поступивших от мыши. Компьютер посылает эту команду, если переданные от мыши данные имеют недопустимое значение. Получив ее, мышь выполняет повторную передачу последнего пакета данных. Если данные опять ошибочны, компьютер может повторить запрос командой `Resend` либо выполнить программный сброс командой `Reset`. В любом случае регистр данных (`60h`) будет содержать значение `FAh`.

2.3.3. Команда *Set Defaults* (*F6h*)

Команда позволяет установить все параметры мыши, используемые по умолчанию. После записи в регистр данных кода `FAh` мышь установит следующие значения: масштаб — 1 : 1, разрешение — 4 отсчета на миллиметр, частота дискретизации — 100 выборок и потоковый режим.

2.3.4. Команда *Disable* (*F5h*)

Данная команда позволяет отключить обмен данных между мышью и компьютером в потоковом режиме. Кроме этого, все счетчики движения сбрасываются в нулевое состояние. При этом все другие операции могут осуществляться. Выполнение команды подтверждается записью в регистр данных кода `FAh`.

2.3.5. Команда *Enable* (*F4h*)

Команда позволяет продолжить передачу данных между мышью и компьютером, если установлен потоковый режим. Выполнение команды подтверждается записью в регистр данных кода `FAh`.

2.3.6. Команда *Set Sample Rate* (*F3h*)

Команда позволяет установить частоту дискретизации для мыши PS/2. Команда является двухбайтовой. Вначале в регистр данных записывается код команды F3h. Мышь подтверждает получение команды записью в регистр данных кода FAh. После этого в регистр данных записывается второй байт команды, который содержит значение частоты дискретизации. Мышь вторично подтверждает получение, записав в регистр данных код FAh. Возможные значения частоты дискретизации перечислены в табл. 2.6. Для лучшего понимания работы с данной командой приведу пример ее использования в листинге 2.17.

Листинг 2.17. Установка частоты дискретизации для мыши PS/2

```
@repeat:
in  AL, 64h ; получаем состояние порта
and AL, 10b ; проверяем наличие данных во входном буфере
jnz @repeat ; буфер занят, пытаемся повторить запрос
mov AL, 0D4h; запишем команду передачи байта
out 64h, AL ; запишем значение в командный регистр
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
mov AL, F3h ; запишем код команды Set Sample Rate
out 60h, AL ; в регистр данных
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
@repeat2:
in  AL, 64h ; получаем состояние порта
and AL, 10b ; проверяем наличие данных во входном буфере
jnz @repeat2; буфер занят, пытаемся повторить запрос
mov AL, 0D4h; запишем команду передачи байта
out 64h, AL ; запишем значение в командный регистр
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
mov AL, 64h ; запишем новое значение частоты 100 выборок
out 60h, AL ; в регистр данных
in  AL, 64h ; проверяем регистр состояния
and AL, 20h ; успешно ли передана команда
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
in  AL, 60h ; проверяем наличие кода FAh
cmp AL, 0FAh; в регистре данных
jnz ErrorHnd; если нет, передаем управление обработчику ошибок
```

Этот же код для C++ представлен в листинге 2.18.

Листинг 2.18. Установка частоты дискретизации для мыши PS/2 в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду передачи данных
outPort ( 0x64, 0xD4, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду Set Sample Rate
outPort ( 0x60, 0xF3, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду передачи данных
outPort ( 0x64, 0xD4, 1);
iTimeWait = 50000;
```

```
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт новое значение частоты 100 выборок
outPort ( 0x60, 0x64, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// проверяем наличие кода FAh в регистре данных
inPort ( 0x60, &dwResult, 1);
if ( dwResult != 0xFA)
       return MY_ERROR;
// новая частота дискретизации 100 выборок в секунду установлена
```

2.3.7. Команда *Read Device Type* (*F2h*)

Команда позволяет получить идентификатор устройства. Для мыши PS/2 это значение всегда равно 00h. Мышь подтверждает получение команды записью в регистр данных кода FAh.

2.3.8. Команда *Set Remote Mode* (*F0h*)

Данная команда позволяет включить дистанционный режим управления мышью PS/2. Мышь подтверждает получение команды записью в регистр данных кода FAh. В этом режиме мышь передает данные на компьютер только после подачи последним запроса чтения с помощью команды EBh.

2.3.9. Команда *Set Wrap Mode* (*EEh*)

Команда позволяет установить эхо-режим для мыши PS/2. Мышь подтверждает получение команды записью в регистр данных кода FAh. В этом ре-

жиме любой байт, кроме FFh (Reset) и ECh (Reset Wrap Mode), будет возвращен устройством мыши на компьютер.

2.3.10. Команда *Reset Wrap Mode* (*ECh*)

Команда позволяет выйти из эхо-режима и восстановить прежний режим. Если передача данных в потоковом режиме была заблокирована командой Disable, то прежний режим не будет восстановлен. Кроме того, выполнение данной команды в любом другом режиме не будет иметь никакого результата. Мышь подтверждает получение команды записью в регистр данных кода FAh.

2.3.11. Команда *Read Data* (*EBh*)

Команда позволяет получить текущие значения счетчиков перемещения (по осям X и Y), а также информацию о состоянии кнопок. После успешной передачи данных счетчики перемещения сбрасываются в ноль. Мышь подтверждает получение команды записью в регистр данных кода FAh. Если мышь находится в дистанционном режиме, выполнение данной команды является единственным способом получения данных.

2.3.12. Команда *Set Stream Mode* (*EAh*)

Эта команда позволяет активизировать потоковый режим передачи данных для мыши PS/2. Мышь подтверждает получение команды записью в регистр данных кода FAh. В этом режиме данные от мыши поступают при любом движении или изменении состояния кнопок.

2.3.13. Команда *Status Request* (*E9h*)

Команда позволяет получить пакет данных (три байта) о текущем состоянии счетчиков движения и кнопок мыши PS/2. Формат данных представлен в табл. 2.15. Мышь подтверждает получение команды записью в регистр данных кода FAh. Например, если выполнить команду Reset, то Status Request возвратит 4 байта в следующей очередности: FAh, 00h, 02h и 64h. Первый байт указывает на завершение команды, второй байт — масштаб 1 : 1, принятый по умолчанию, третий байт — разрешение 4 отсчета на миллиметр, и четвертый байт — частоту дискретизации 100 выборок в секунду.

В табл. 2.15 введены следующие обозначения:

- Правая кнопка. Определяет состояние правой кнопки мыши: 1 — кнопка нажата, 0 — кнопка отпущена.
- Средняя кнопка. Определяет состояние средней кнопки мыши: 1 — кнопка нажата, 0 — кнопка отпущена.

- Левая кнопка. Определяет состояние левой кнопки мыши: 1 — кнопка нажата, 0 — кнопка отпущена.
- Масштаб. Определяет текущий режим масштабирования: 1 — режим 2 : 1, 0 — 1 : 1.
- Передача. Определяет передачу данных для потокового режима: 1 — передача разрешена, 0 — передача запрещена.
- Режим. Определяет текущий режим работы мыши: 1 — дистанционный, 0 — потоковый.
- Разрешение. Определяет текущее разрешение мыши. Может принимать значение от `0h` до `3h`.
- Частота дискретизации. Определяет текущую частоту дискретизации мыши. Может принимать значение от 10 до 200.

Таблица 2.15. *Формат данных состояния мыши*

<table>
<tr><th>Бит
Байт</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>1</td><td>0</td><td>Режим</td><td>Передача</td><td>Масштаб</td><td>0</td><td>Левая кнопка</td><td>Средняя кнопка</td><td>Правая кнопка</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="2">Разрешение</td></tr>
<tr><td>3</td><td colspan="8">Частота дискретизации</td></tr>
</table>

Для наглядности приведу простой пример использования команды `Status Request` в листинге 2.19.

Листинг 2.19. Использование команды Status Request

```
stat_1 db ? ; переменная для хранения первого байта состояния
stat_2 db ? ; переменная для хранения второго байта состояния
stat_3 db ? ; переменная для хранения третьего байта состояния
...         ; общий код программы
@repeat:
in  AL, 64h  ; получаем состояние порта
and AL, 10b  ; проверяем наличие данных во входном буфере
jnz @repeat  ; буфер занят, пытаемся повторить запрос
mov AL, 0D4h ; запишем команду передачи байта
out 64h, AL  ; запишем значение в командный регистр
in  AL, 64h  ; проверяем регистр состояния
and AL, 20h  ; успешно ли передана команда
jnz ErrorHnd ; если нет, передаем управление обработчику ошибок
```

```
mov AL, 0E9h ; запишем код команды Status Request
out 60h, AL  ; в регистр данных
@repeat2:
in  AL, 64h  ; получаем состояние порта
and AL, 20h  ; проверяем данные
jnz @repeat2 ; если данных нет, повторяем
in  AL, 60h  ; получаем код завершения FAh
cmp AL, 0FAh ; если код отсутствует
jnz ErrorHnd ; передаем управление обработчику ошибок
@repeat3:
in  AL, 64h  ; получаем состояние порта
and AL, 20h  ; проверяем данные
jnz @repeat3 ; если данных нет, повторяем
in  AL, 60h  ; получаем первый байт состояния
mov stat_1, AL; сохраняем в переменной stat_1
@repeat4:
in  AL, 64h  ; получаем состояние порта
and AL, 20h  ; проверяем данные
jnz @repeat4 ; если данных нет, повторяем
in  AL, 60h  ; получаем второй байт состояния
mov stat_2, AL; сохраняем в переменной stat_2
@repeat5:
in  AL, 64h  ; получаем состояние порта
and AL, 20h  ; проверяем данные
jnz @repeat5 ; если данных нет, повторяем
in  AL, 60h  ; получаем третий байт состояния
mov stat_3, AL; сохраняем в переменной stat_3
```

Код упрощен для того, чтобы можно было легче понять принцип работы с управляющими командами. Например, циклы желательно делать конечными, чтобы программа не зависла при отсутствии данных. Кроме того, может потребоваться запрещение прерывания мыши (смотрите описание регистра команд) перед вызовом управляющих команд, а после завершения работы — разрешение генерации прерываний.

Аналогичный код получения состояния мыши для C++ показан в листинге 2.20.

Листинг 2.20. Использование команды Status Request в C++

```
// переменная для хранения результата
DWORD dwResult  = 0;
// переменные для хранения трех байтов текущего состояния мыши
BYTE bStaus_1 = 0, bStaus_2 = 0, bStaus_3 = 0
int iTimeWait = 50000;
```

```
// проверяем наличие данных во входном буфере
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду передачи данных
outPort ( 0x64, 0xD4, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт код команды Status Request
outPort ( 0x60, 0xE9, 1);
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x20) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// проверяем код выполнения команды FAh
inPort ( 0x60, &dwResult, 1);
if ( dwResult != 0xFA)
       return MY_ERROR;
// ожидаем данные
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
```

```
        if ( (dwResult & 0x20) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// получаем первый байт состояния
inPort ( 0x60, &dwResult, 1);
// сохраняем значение первого байта в переменной
bStaus_1 = (BYTE) dwResult;
// ожидаем данные
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x20) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// получаем второй байт состояния
inPort ( 0x60, &dwResult, 1);
// сохраняем значение второго байта в переменной
bStaus_2 = (BYTE) dwResult;
// ожидаем данные
iTimeWait = 50000;
// проверяем, успешно ли передана команда
while ( -- iTimeWait > 0)
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x20) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// получаем третий байт состояния
inPort ( 0x60, &dwResult, 1);
// сохраняем значение третьего байта в переменной
bStaus_3 = (BYTE) dwResult;
```

2.3.14. Команда *Set Resolution* (*E8h*)

Данная команда предназначена для установки разрешения мыши PS/2. Команда является двухбайтовой. Вначале в регистр данных записывается

код команды E8h. Мышь подтверждает получение команды записью в регистр данных кода FAh. После этого в регистр данных записывается второй байт команды, который содержит новое значение разрешения. Мышь вторично подтверждает получение, записав в регистр данных код FAh. Возможные значения кода разрешения перечислены в табл. 2.7.

2.3.15. Команда *Set Scaling 2:1* (*E7h*)

Команда предназначена для установки режима масштабирования 2:1, позволяющего улучшить чувствительность мыши за счет нелинейного пересчета датчиков движения (см. табл. 2.3). Мышь подтверждает получение команды записью в регистр данных кода FAh.

2.3.16. Команда *Set Scaling 1:1* (*E6h*)

Эта команда позволяет восстановить режим масштабирования, используемый по умолчанию и равный 1:1. Мышь подтверждает получение команды записью в регистр данных кода FAh.

* * *

На этом можно завершить описание команд управления мышью PS/2. Хочу только заметить, что в своих программах желательно проверять различные ошибки (см. табл. 2.13) завершения, а не только код FAh. Это позволит точнее определить проблему сбоя и принять соответствующие меры.

Теперь рассмотрим, какие возможности для поддержки устройства мыши предлагает нам фирма Microsoft.

2.4. Использование Win32 API

Набор возможностей, предоставляемый этим программным интерфейсом, небогат, но позволяет решить основные вопросы использования мыши в операционных системах Windows. Общие темы, которые мы рассмотрим здесь, можно свести к короткому списку:

1. Настройка мыши.
2. Работа с курсором.

2.4.1. Настройка мыши

Удобство работы с мышью в программе часто служит определяющим фактором, по которому судят о качестве программного обеспечения. Особенно это касается пользователей, использующих левую руку в качестве основной, проще говоря, левшей. Мало кто из разработчиков программного обеспече-

ния думает об этом. Конечно, в настройках Windows есть возможность поменять местами левую и правую кнопки мыши, но у рядового пользователя в связи с этим возникают проблемы. Во-первых, нужно хорошо знать систему, чтобы найти нужные настройки. Во-вторых, такую операцию приходится выполнять достаточно часто, что отвлекает от основной работы на компьютере. Было бы гораздо проще, если бы каждый разработчик программного обеспечения думал об этом заранее и добавлял в свою программу возможность менять значения левой и правой кнопок мыши. Это добавит программе только плюс.

Программно решить эту задачу в Windows не сложно. Существует два способа, о которых я хотел бы рассказать. Прежде всего, замечу, что оба способа изменяют глобальные настройки системы, поэтому следует восстанавливать стандартные значения перед закрытием своей программы. Первый способ состоит из вызова функции SwapMouseButton. Данная функция имеет всего один аргумент (тип BOOL), который определяет направление операции. Чтобы поменять местами левую и правую кнопки мыши, присвойте аргументу значение TRUE. Для восстановления значения по умолчанию вызовите функцию со значением FALSE. Функция SwapMouseButton возвратит нулевое значение, если до этого назначение кнопок не менялось, и единицу в обратном случае. Второй способ состоит в использовании универсальной функции SystemParametersInfo с установленной опцией SPI_SETMOUSEBUTTONSWAP.

Для лучшего понимания просмотрите примеры, представленные в листингах 2.21 и 2.22.

Листинг 2.21. Использование функции SwapMouseButton

```
// напишем свою функцию для смены кнопок мыши
bool LeftToRight ( bool bLR)
{
       if ( SwapMouseButton ( bLR) != 0)
       // состояние кнопок уже изменено
       return false;
return true;
}
// используем нашу функцию в коде
//  ...
void MouseButtons ()
{
       if ( !LeftToRight ( true))
       // если состояние кнопок уже менялось, восстановив по умолчанию
              LeftToRight ( false);
}
```

Для того чтобы узнать текущее состояние кнопок, можно воспользоваться функцией GetSystemMetrics с аргументом SM_SWAPBUTTON. Если значения левой и правой кнопок были изменены, функция возвратит ненулевое значение, иначе результат будет равен нулю.

Листинг 2.22. Использование функции SystemParametersInfo

```
// напишем свою функцию для смены кнопок мыши
bool LeftToRight ( bool bLR)
{
   SystemParametersInfo ( SPI_SETMOUSEBUTTONSWAP, (UINT) bLR, NULL,
                          SPIF_UPDATEINIFILE);
}
```

Вызов функции SystemParametersInfo записывает в системный файл WIN.INI значение в виде цифры (например, "1"). Однако при последующей загрузке Windows параметры мыши могут остаться прежними, независимо от ваших действий. Происходит это потому, что панель управления тоже записывает в указанный системный файл информацию о состоянии кнопок мыши, но она использует буквенное обозначение (например, вместо "1" слово "Yes"). Приоритет всегда будет отдан буквенному значению ("Yes" или "No"), поэтому, чтобы избежать таких неожиданностей, следует использовать функцию SwapMouseButton.

Кроме изменения состояния левой и правой кнопок мыши, можно программно менять время реакции системы на двойной щелчок. Для этого используется функция SetDoubleClickTime, имеющая всего один аргумент, который задает максимальное значение времени ожидания (в миллисекундах) между первым и вторым щелчком мыши. Значение по умолчанию, используемое в Windows, равно 500 мс. Для установки времени по умолчанию достаточно вызвать функцию SetDoubleClickTime с нулевым значением аргумента. В случае ошибки функция возвращает 0, и для получения детального описания возникшей ситуации следует вызвать GetLastError.

Кроме SetDoubleClickTime, существует функция GetDoubleClickTime, которая позволяет узнать текущее значение времени реакции на двойной щелчок. Она не имеет аргументов и возвращает тип UINT, характеризующий время реакции системы на двойной щелчок мыши (в миллисекундах). В листинге 2.23 приведен пример работы с функциями SetDoubleClickTime и GetDoubleClickTime.

Листинг 2.23. Использование функций SetDoubleClickTime и GetDoubleClickTime

```
// напишем собственную функцию установки времени двойного щелчка мыши
bool SetDblClickTime ( unsigned int uTime)
```

```
{
   unsigned int uCurrentTime = 0;
   // получаем текущее значение
   uCurrentTime = GetDoubleClickTime ();
   // если оно совпадает с устанавливаемым, выходим из функции
   if ( uTime == uCurrentTime)
       return false;
   // если нет, устанавливаем новое значение
   if ( SetDoubleClickTime ( uTime) == 0)
      {
         // произошла ошибка, вызываем GetLastError, если нужно
         return false;
       }
return true;
}
```

Как видно из примеров, работа с этими функциями не представляет никаких проблем. Хочу только заметить, что изменение времени реакции между двумя щелчками мышью действует для всей операционной системы.

Кроме описанного способа, существует еще один, использующий универсальную функцию `SystemParametersInfo`. Для установки текущего значения времени применяется параметр `SPI_SETDOUBLECLICKTIME`.

Существует еще один параметр, которым можно управлять программно, — скорость движения указателя мыши по экрану. Для реализации этой возможности также применяется функция `SystemParametersInfo`. Получить текущее значение скорости можно, вызвав ее с параметром `SPI_GETMOUSESPEED`, а установить новое значение — с параметром `SPI_SETMOUSESPEED`. Диапазон возможных значений лежит в пределах от 1 до 20. По умолчанию используется значение 10. Пример функции, изменяющей скорость движения указателя, представлен в листинге 2.24.

Листинг 2.24. Установка скорости движения указателя мыши

```
// напишем свою функцию для управления скоростью движения указателя мыши
int SpeedMouse ( int iSpeed, bool bFlag)
{
   if ( bFlag) // установить скорость
     {
       if ( ( iSpeed > 0) && ( iSpeed < 21)) // проверяем диапазон
        {
          SystemParametersInfo ( SPI_SETMOUSESPEED, 0, (PVOID) iSpeed,
                                      SPIF_SENDCHANGE | SPIF_UPDATEINIFILE);
        }
```

```
   else // получаем текущее значение скорости
     {
       int iCurrentSpeed = 10; // значение по умолчанию
       SystemParametersInfo ( SPI_GETMOUSESPEED, 0, &iCurrentSpeed, 0);
       return iCurrentSpeed;
     }
return 0;
}
```

Представленная читателям функция `SpeedMouse` имеет два аргумента. Первый аргумент передает значение скорости, а второй указывает на тип выполняемой операции (установка или получение текущего значения скорости).

Последняя возможность управления мышью, которую я хотел бы рассмотреть, представляет собой блокировку всех событий мыши. Функция называется `BlockInput` и позволяет блокировать не только мышь, но и клавиатуру. Единственный аргумент функции, установленный в `TRUE`, блокирует мышь и клавиатуру. В случае ошибки функция возвращает 0, и для получения детального описания возникшей ситуации следует вызвать `GetLastError`. Пример работы данной функции приведен в листинге 2.25.

Листинг 2.25. Блокировка мыши и клавиатуры

```
// обязательно включите файл Winable.h
#include <Winable.h>
// пишем функцию блокировки
bool LockMouse ( bool Action)
{
    if ( BlockInput ( Action) == 0)
     {
        // ошибка выполнения
        return false;
     }
return true;
}
```

Прежде чем использовать функцию `BlockInput`, следует учесть несколько важных факторов:

1. Функция полностью блокирует мышь и клавиатуру, поэтому в программе следует задавать время по таймеру, после которого устройства будут разблокированы.
2. Восстановить доступ к устройствам можно с помощью перезагрузки компьютера или нажав комбинацию клавиш <Ctrl>+<Alt>+<Del>.

Из сказанного выше можно сделать основной вывод: разработчик программы, использующий функцию BlockInput, должен позаботиться о полном восстановлении работы мыши и клавиатуры. В процессе блокировки программист может применять функцию SendInput, имитирующую нажатия кнопок мыши и клавиатуры.

И последняя возможность, о которой хотелось бы рассказать, — это получение краткой информации о мыши. Чтобы определить, установлена ли в системе мышь, можно вызвать функцию GetSystemMetrics со значением SM_MOUSEPRESENT. Если мышь установлена, функция возвратит ненулевое значение, иначе будет возвращен 0. В листинге 2.26 приведен пример использования этой функции.

Листинг 2.26. Проверка наличия в системе устройства мыши

```
// пишем функцию для проверки подключения мыши
bool IsMouse ()
{
    if ( GetSystemMetrics ( SM_MOUSEPRESENT))
        return true; // мышь установлена
return false;
}
```

Кроме определения наличия в системе мыши, можно получить данные о количестве кнопок и наличии колеса прокрутки. Для получения количества кнопок мыши нужно вызвать функцию GetSystemMetrics со значением SM_CMOUSEBUTTONS, а для проверки наличия колесика — со значением SM_MOUSEWHEELPRESENT. В первом случае функция вернет количество кнопок или значение 0, если мышь отсутствует. Во втором случае функция возвратит ненулевое значение, если колесо прокрутки имеется. В листинге 2.27 показано, как это делается.

Листинг 2.27. Получение информации о числе кнопок и наличии колеса прокрутки

```
// напишем функцию, определяющую число кнопок мыши
int GetMouseButtons ()
{
    int iNum = 0;
    iNum = GetSystemMetrics (SM_CMOUSEBUTTONS);
    if ( iNum)
        return iNum; // количество кнопок
return iNum; // мышь отсутствует
}
```

```
// напишем функцию для определения колеса прокрутки
bool IsMouseWheel ()
{
if ( GetSystemMetrics ( SM_MOUSEWHEELPRESENT))
        return true; // мышь установлена
return false;
}
```

2.4.2. Работа с курсором

В Windows курсор мыши представляет собой черно-белый или цветной значок и может быть статическим или динамическим (так называемый "живой" указатель). В системе всегда имеется стандартный набор курсоров для использования в различных ситуациях. Этот набор называется системным и может применяться в программе по умолчанию. Мы рассмотрим наиболее часто востребованную возможность работы с курсорами для создания визуального ожидания завершения какой-либо операции в программе. Те читатели, кто знаком с классом `CWaitCursor` из библиотеки MFC, сразу поймут, о чем идет речь. В каждой программе так или иначе встречаются функции, требующие определенного времени для завершения. Чтобы пользователь видел, что программа не "зависла", а продолжает выполняться, необходимо как-то сообщить ему это. Обычно текущий курсор (как правило, в виде стрелки) на время выполнения операции заменяется другим (как правило, в виде песочных часов). Для реализации такой возможности мы создадим полноценный класс, который можно будет легко использовать в различных программах. Назовем наш класс `CWaitMouse`. Файл объявлений (CWaitMouse.h) представлен в листинге 2.28, а файл реализации (CWaitMouse.cpp) — в листинге 2.29.

Листинг 2.28. Файл CWaitMouse.h

```
class CWaitMouse
{
public:
       CWaitMouse ();      // конструктор по умолчанию
       ~CWaitMouse () { } // пустой деструктор
       // общедоступные функции
       void Wait ();     // функция отображает курсор ожидания
       void Restore (); // функция восстанавливает прежний курсор
private:
      bool bStatus; // хранит текущее состояние курсора
      void SetWait ( bool bWait); // установка курсора ожидания
};
```

Листинг 2.29. Файл CWaitMouse.cpp

```
#include " CWaitMouse.h"
// прямо в конструкторе изменяем вид курсора
CWaitMouse :: CWaitMouse ()
{
      bStatus = false; // инициализация
      SetWait ( true); // устанавливаем курсор ожидания
}
// функция восстанавливает прежний вид курсора
void CWaitMouse :: Restore()
{
       SetWait ( false);
}
// функция устанавливает курсор ожидания
void CWaitMouse :: Wait ()
{
       // проверяем, не установлен ли курсор ожидания ранее
       if ( bStatus) return;
       // если нет, устанавливаем
       SetWait ( true);
}
// функция обработки курсора
void CwaitMouse :: SetWait (bool bWait)
{
      // дескриптор для нового курсора
      HCURSOR hCursor = NULL;
      // дескриптор для хранения старого курсора
      static HCURSOR hStarikCursor = NULL;
      // проверяем тип операции
      if ( bWait) // показать курсор ожидания
       {
          hCursor = LoadCursor (NULL, IDC_WAIT);
          hStarikCursor = SetCursor ( hCursor);
          bStatus = true;
       }
      else // восстановить прежний курсор мыши
       {
          SetCursor ( hStarikCursor);
          bStatus = false;
       }
} // конец функции
```

Внимательно изучив код нашего класса, вы увидите, что вся основная работа выполняется в функции `SetWait`. Для загрузки курсора ожидания мы вы-

полнили два основных действия: загрузили стандартный курсор ожидания (IDC_WAIT) и заменили им текущий курсор. Загрузка курсора выполнена с помощью библиотечной функции LoadCursor. Данная функция имеет два параметра: первый указывает на дескриптор нашей программы (тип HINSTANCE), а второй определяет тип курсора для мыши. Поскольку мы используем стандартный курсор, первый аргумент можно установить в NULL. Второму аргументу присваиваем одно из значений, перечисленных в табл. 2.16.

Таблица 2.16. *Возможные значения для системных курсоров*

Константа	Описание
IDC_WAIT	Значок "Песочные часы"
IDC_ARROW	Значок стандартной стрелки
IDC_CROSS	Значок перекрестья
IDC_HELP	Значок в виде вопросительного знака со стрелкой
IDC_HAND	Значок в виде руки
IDC_UPARROW	Значок в виде вертикальной стрелки
IDC_SIZEALL	Значок изменения размеров

После успешного завершения функция LoadCursor вернет дескриптор загруженного курсора (тип HCURSOR). Для вывода нового курсора на экран необходимо вызвать функцию SetCursor. Она имеет один аргумент, куда заносится дескриптор необходимого курсора. В случае успешного завершения функция вернет дескриптор старого курсора, который мы сохраняем в статической переменной для последующего восстановления. Вот и все премудрости.

Существует возможность программно скрыть курсор с экрана. Для этого применяется функция ShowCursor. Функция имеет всего один аргумент. Если нужно скрыть курсор, вызовите эту функцию с аргументом FALSE. Хитрость состоит в том, что функция при каждом вызове увеличивает (TRUE) или уменьшает (FALSE) внутренний системный счетчик. Когда значение счетчика больше или равно 0, курсор видим на экране. Если мышь присутствует, начальное значение счетчика равно 0, иначе — 1. Исходя из вышесказанного, необходимо следить, чтобы каждый вызов ShowCursor имел повторный вызов этой функции в программе. После выполнения функция возвращает текущее значение счетчика. Примеры работы с данной функцией я приводить не буду, поскольку она достаточно проста в использовании.

На этом мне бы хотелось завершить тему программирования мыши и перейти к не менее важному устройству — клавиатуре.

ГЛАВА 3

Клавиатура

Клавиатура является одним из самых старых, но не менее актуальных и сегодня устройств. Без нее трудно представить полноценный компьютер. Несмотря на бурное развитие программных средств голосового управления компьютером, клавиатура остается наиболее надежным и совершенным средством управления программным обеспечением. Важность ее трудно переоценить, что позволяет мне утверждать о необходимости изучения различных способов программирования этого привычного для каждого пользователя устройства.

Мы рассмотрим три основных способа программирования клавиатуры:

1. Поддержка клавиатуры посредством функций BIOS.
2. Работа с контроллером клавиатуры напрямую через порты.
3. Программирование клавиатуры в Win32 API.

Каждый программист может выбрать один из трех вариантов, в зависимости от поставленной задачи, но, прежде чем мы приступим к рассмотрению этих способов, позвольте мне немного рассказать о самом устройстве клавиатуры.

3.1. Общие сведения

На сегодняшний день существует два основных типа клавиатуры: AT и PS/2. Первый тип уже морально устарел и практически полностью вытеснен современным стандартом PS/2. Кроме основных типов клавиатур существуют и другие, реже встречающиеся устройства: USB и инфракрасные. Для поддержки клавиатуры используется интегрированный в системный чип контроллер (например, Intel 8042 или VIA 8242), одновременно поддерживающий и мышь PS/2.



По количеству клавиш клавиатуры можно разделить на несколько типов:

- клавиатуры XT с 83 клавишами. Появились в 1981 году. Использовали 5-штырьковый разъем DIN. Передача данных была организована по однонаправленному последовательному протоколу. На данный момент полностью устарели;
- клавиатуры AT с 84—101 клавишами. Появились в 1984 году. Используют 5-штырьковый разъем DIN. Передача данных организована по двунаправленному последовательному протоколу;
- клавиатуры PS/2 с 84—101 клавишами. Появились в 1987 году. Используют 5-штырьковый разъем mini-DIN. Передача данных организована по двунаправленному последовательному протоколу;
- современные клавиатуры PS/2 с 101—104 (или более) клавишами. Используют 6-штырьковый разъем mini-DIN. Передача данных организована по двунаправленному последовательному протоколу.

Как правило, все клавиатуры состоят из набора клавиш и встроенного микропроцессора. Главная задача микропроцессора состоит в передаче скан-кода нажатой (отпущенной) клавиши (комбинации клавиш) на компьютер и обработку управляющих команд от него. Для ускорения работы имеется встроенный буфер (обычно 16 байт), позволяющий хранить данные обмена между клавиатурой и компьютером. Для управления клавиатурой разработан набор специальных управляющих команд. Следует иметь в виду, что некоторые современные PS/2 клавиатуры могут не поддерживать отдельные команды. Команды управления будут рассмотрены далее в этой главе. Обмен данными между клавиатурой и компьютером (контроллером клавиатуры, расположенном на материнской плате) работает по протоколу, разработанному фирмой IBM. Встроенный в клавиатуру микропроцессор сканирует шину в ожидании нажатия (отпускания) любой клавиши. Если произошло нажатие (отпускание или удержание) клавиши, вырабатывается определенный скан-код (набор скан-кодов), позволяющий компьютеру определить текущее состояние клавиатуры. Каждая клавиша имеет свой уникальный скан-код. Описание некоторых скан-кодов клавиш приводятся далее в табл. 3.2 и 3.3. Каждый переданный компьютеру скан-код (числовое значение) обрабатывается и преобразовывается в код ASCII, который и применяется для передачи смыслового содержания нажатой клавиши. Скан-код для стандартной клавиатуры (84 клавиши) имеет размер 1 байт, а для расширенной — от 2 до 4 байтов. Чтобы отличить расширенный скан-код от обычного, в качестве первого байта всегда выступает значение `E0h` (например, код левой клавиши <Alt> равен `38h`, а правой — `E0h,38h`). Кроме уникального кода нажатия, каждая клавиша имеет свой код отпускания. Как правило, этот код состоит из двух байт, первый из которых всегда равен `F0h`. На расширенных клавиатурах коды отпускания имеют размер три байта, где первые два байта всегда равны `E0h,F0h`, а третий байт является последним байтом скан-кода нажатия.

Существует три основных набора скан-кодов: стандартный XT (поддерживается некоторыми современными клавиатурами), набор по умолчанию (поддерживается всеми современными клавиатурами) и набор кодов PS/2 (необязательный и может не поддерживаться современными клавиатурами). Далее в табл. 3.17 приводится описание второго набора скан-кодов. В любом случае рекомендую читателям скачать в Интернете и распечатать для себя все скан-коды, поддерживаемые современными клавиатурами (особенно мультимедийными).

3.2. Использование функций BIOS

Данный способ базируется на стандартном наборе функций BIOS, использующих прерывание `int 16h`. Список всех функций представлен в табл. 3.1. Современные системы поддерживают следующие типы клавиатур: 84-клавишные, 102-клавишные и 122-клавишные. Каждая клавиша имеет свой скан-код, который обрабатывается BIOS при нажатии или отпускании. Для поддержки 84-клавишных устройств используются только функции `00h`, `01h` и `02h`. Функции `10h`, `11h` и `12h` поддерживают 83-клавишные и 102-клавишные устройства. Функции `20h`, `21h` и `22h` поддерживают все типы клавиатур.

Таблица 3.1. *Список функций BIOS*

Код функции	Описание
00h	Получить скан-код и ASCII-код клавиши
01h	Проверить, была ли нажата клавиша
02h	Получить состояние специальных клавиш
03h	Управление режимом автоповтора и значением задержки
04h	Использовать звуковой сигнал
05h	Сохранить код клавиши в буфере клавиатуры
09h	Получить информацию о возможностях клавиатуры
0Ah	Получить идентификатор клавиатуры
10h	Прочитать код клавиши для расширенной клавиатуры
11h	Проверить, была ли нажата клавиша на расширенной клавиатуре
12h	Получить состояние специальных клавиш на расширенной клавиатуре
20h	Получить скан-код и ASCII-код клавиши 122-клавишной клавиатуры
21h	Проверить, была ли нажата клавиша на 122-клавишной клавиатуре

Таблица 3.1 (окончание)

Код функции	Описание
22h	Получить состояние специальных клавиш 122-клавишной клавиатуры
FFh	Добавить код клавиши в конец буфера клавиатуры

Рассмотрим подробнее каждую функцию BIOS.

3.2.1. Функция *00h*

Функция позволяет получить скан-код и ASCII-код нажатой клавиши. С помощью данной функции нельзя получить код дополнительных клавиш на расширенной клавиатуре. Для этого следует использовать функцию 10h. Функция считывает символ из буфера клавиатуры, а затем очищает буфер. При получении кодов от управляющих клавиш (например, <Пробел>) в регистр AL будет записан ASCII-код, равный 0.

Использование:

1. В регистр AH следует поместить код функции 00h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции в регистр AH будет помещен скан-код BIOS символа, а в регистр AL — ASCII-код символа. Стандартный набор значений скан-кодов для клавиатуры представлен в табл. 3.2, а расширенный — в табл. 3.3. Кроме того, в табл. 3.4 представлены коды управляющих символов ASCII.

Таблица 3.2. Скан-коды клавиатуры

Клавиша	Код	Клавиша	Код	Клавиша	Код
Esc	01h	Enter	1Ch	*	37h
1 и !	02h	Ctrl	1Dh	Alt	38h
2 и @	03h	A	1Eh	Space	39h
3 и #	04h	S	1Fh	Caps Lock	3Ah
4 и $	05h	D	20h	F1	3Bh
5 и %	06h	F	21h	F2	3Ch
6 и ^	07h	G	22h	F3	3Dh
7 и &	08h	H	23h	F4	3Eh

Таблица 3.2 *(окончание)*

Клавиша	Код	Клавиша	Код	Клавиша	Код
8 и *	09h	J	24h	F5	3Fh
9 и (	0Ah	K	25h	F6	40h
0 и)	0Bh	L	26h	F7	41h
- и _	0Ch	; и :	27h	F8	42h
= и +	0Dh	‘ и "	28h	F9	43h
Back Space	0Eh	‘ и ~	29h	F10	44h
Tab	0Fh	Left Shift	2Ah	Num Lock	45h
Q	10h	\ и |	2Bh	Scroll Lock	46h
W	11h	Z	2Ch	Home	47h
E	12h	X	2Dh	Arrow Up	48h
R	13h	C	2Eh	Page Up	49h
T	14h	V	2Fh	—	4Ah
Y	15h	B	30h	Arrow Left	4Bh
U	16h	N	31h	5	4Ch
I	17h	M	32h	Arrow Right	4Dh
O	18h	, и <	33h	+	4Eh
P	19h	. и >	34h	End	4Fh
[и {	1Ah	/ и ?	35h	?	50h
] и }	1Bh	Right Shift	36h	Page Down	51h
Insert	52h	Delete	53h		

Таблица 3.3. *Коды для расширенной клавиатуры*

Клавиша	Код	Клавиша	Код	Клавиша	Код	Клавиша	Код
F1	3Bh	Alt + F1	68h	Shift + F1	54h	Ctrl + F1	5Eh
F2	3Ch	Alt + F2	69h	Shift + F2	55h	Ctrl + F2	5Fh
F3	3Dh	Alt + F3	6Ah	Shift + F3	56h	Ctrl + F3	60h
F4	3Eh	Alt + F4	6Bh	Shift + F4	57h	Ctrl + F4	61h
F5	3Fh	Alt + F5	6Ch	Shift + F5	58h	Ctrl + F5	62h

Таблица 3.3 (окончание)

Клавиша	Код	Клавиша	Код	Клавиша	Код	Клавиша	Код
F6	40h	Alt + F6	6Dh	Shift + F6	59h	Ctrl + F6	63h
F7	41h	Alt + F7	6Eh	Shift + F7	5Ah	Ctrl + F7	64h
F8	42h	Alt + F8	6Fh	Shift + F8	5Bh	Ctrl + F8	65h
F9	43h	Alt + F9	70h	Shift + F9	5Ch	Ctrl + F9	66h
F10	44h	Alt + F10	71h	Shift + F10	5Dh	Ctrl + F10	67h
F11	85h	Alt + F11	8Bh	Shift + F11	87h	Ctrl + F11	89h
F12	86h	Alt + F12	8Ch	Shift + F12	88h	Ctrl + F12	8Ah

Таблица 3.4. Управляющие символы ASCII

Код	Название	Описание
0	NUL	Конец строки или пустой символ
1	SOH	Начало заголовка
2	STX	Начало текста
3	ETX	Конец текста
4	EOT	Конец передачи
5	ENQ	Подтверждающий запрос
6	ACK	Подтверждение
7	BEL	Звуковой сигнал
8	BS	Возврат на одну позицию влево
9	HT	Горизонтальная табуляция
0A	LF	Перевод строки
0B	VT	Вертикальная табуляция
0C	FF	Переход на новую страницу
0D	CR	Возврат каретки
0E	SO	Нижний регистр
0F	SI	Верхний регистр
10	DLE	Освободить канал связи
11	DC1	Управление устройством 1
12	DC2	Управление устройством 2

Таблица 3.4 (окончание)

Код	Название	Описание
13	DC3	Управление устройством 3
14	DC4	Управление устройством 4
15	NAK	Подтверждение ошибки передачи
16	SYN	Синхронизация
17	ETB	Конец передаваемого блока
18	CAN	Отмена
19	EM	Конец носителя
1A	SUB	Замена
1B	ESC	Выход или переход
1C	FS	Разделитель файлов
1D	GS	Разделитель групп
1E	RS	Разделитель записей
1F	US	Разделитель полей
20	SP	Пробел
7F	DEL	Удаление

Пример работы с функцией 00h, показанный в листинге 3.1, позволяет определить, была ли нажата управляющая клавиша.

Листинг 3.1. Использование функции 00h

```
@repeat:
mov AH, 0     ; код функции 00h
int 16h       ; вызываем прерывание
or  AL, AL    ; проверяем регистр AL
jnz @repeat   ; если нажата не управляющая клавиша, повторяем
mov AL, AH    ; перепишем управляющий код в AL
```

3.2.2. Функция *01h*

Данная функция позволяет проверить нажатие клавиши на клавиатуре. Если клавиша была нажата, функция запишет в соответствующие регистры скан-код и ASCII-код клавиши. Буфер клавиатуры не очищается.

Использование:

1. В регистр AH следует поместить код функции 01h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции в регистр AH будет помещен скан-код BIOS символа, а в регистр AL — ASCII-код символа. Кроме того, если нажатия клавиши не было, флаг ZF будет установлен в 1. Если клавиша была нажата, флаг ZF будет сброшен. Приведу простой пример, в котором проверяется нажатие клавиши <Esc> (листинг. 3.2).

Листинг 3.2. Использование функции 01h

```
@repeat:
mov AH, 1    ; код функции 01h
int 16h      ; вызываем прерывание
jz  @repeat  ; нажатия клавиши не было, повторяем опрос
mov AH, 0    ; клавиша была нажата
int 16h      ; читаем код клавиши
cmp AL, 27   ; сравниваем полученный ASCII-код с кодом ESC (1Bh)
jne @repeat  ; если код не соответствует клавише ESC, повтор
```

3.2.3. Функция *02h*

Эта функция позволяет получить состояние специальных клавиш на клавиатуре: <Shift>, <Ctrl>, <Alt>, <Num Lock>, <Scroll Lock>, <Caps Lock> и <Insert>.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр AL будет хранить байт состояния специальных клавиш. Формат этого байта представлен в табл. 3.5.

Таблица 3.5. *Формат байта состояния клавиатуры*

Бит	Описание
0	Правая клавиша Shift (1 — нажата, 0 — не нажата)
1	Левая клавиша Shift (1 — нажата, 0 — не нажата)
2	Любая клавиша Ctrl (1 — нажата, 0 — не нажата)

Таблица 3.5 (окончание)

Бит	Описание
3	Любая клавиша Alt (1 — нажата, 0 — не нажата)
4	Переключатель Scroll Lock (1 — включен, 0 — выключен)
5	Переключатель Num Lock (1 — включен, 0 — выключен)
6	Переключатель Caps Lock (1 — включен, 0 — выключен)
7	Переключатель Insert (1 — включен, 0 — выключен)

Простой пример, определяющий нажатие клавиши <Alt>, представлен в листинге 3.3.

Листинг 3.3. Использование функции `02h`

```
GetAlt:
mov AH, 2     ; код функции 02h
int 16h       ; вызываем прерывание
and Al, 8     ; проверяем нужный бит
jnz GetAlt    ; ни одна клавиша Alt не нажата, повтор
```

3.2.4. Функция *03h*

Функция позволяет установить режим автоповтора и время задержки.

Использование:

1. В регистр `AH` следует поместить код функции `03h`.
2. В регистр `AL` надо записать тип операции:
   - `00h` — значения по умолчанию (только для некоторых клавиатур PS/2) для автоповтора и времени задержки;
   - `04h` — отключение автоповтора (только для некоторых клавиатур PS/2);
   - `05h` — установка количества повторов и времени задержки для всех стандартных клавиатур;
   - `06h` — получение текущего значения частоты повторений и времени задержки (поддерживается некоторыми новыми клавиатурами PS/2).
3. В регистр `BH` необходимо записать значение кода задержки (табл. 3.6).
4. В регистр `BL` необходимо записать значение кода частоты повторений (табл. 3.7).
5. Вызвать прерывание `int 16h`.

Таблица 3.6. *Код задержки*

Значение кода	Время задержки, мс
00h	250
01h	500
02h	750
03h	1000

Таблица 3.7. *Код частоты повторений*

Значение кода	Частота, символов/с	Значение кода	Частота, символов/с
00h	30,0	10h	7,5
01h	26,7	11h	6,7
02h	24,0	12h	6,0
03h	21,8	13h	5,5
04h	20,0	14h	5,0
05h	18,5	15h	4,6
06h	17,1	16h	4,3
07h	16,0	17h	4,0
08h	15,0	18h	3,7
09h	13,2	19h	3,3
0Ah	12,0	1Ah	3,0
0Bh	10,9	1Bh	2,7
0Ch	10,0	1Ch	2,5
0Dh	9,2	1Dh	2,3
0Eh	8,6	1Eh	2,1
0Fh	8,0	1Fh	2,0

Выход:

После выполнения функции регистр BL будет хранить значение частоты повтора, а регистр BH — время задержки. Выходные параметры истинны только для типа операции с кодом 06h.

Рассмотрим пример работы с функцией 03h (листинг 3.4), в котором попытаемся установить время задержки 750 мс и частоту повтора 20 символов в секунду.

Листинг 3.4. Использование функции 03h

```
mov AH, 3     ; код функции 03h
mov AL, 5     ; тип операции
mov BH, 2     ; время задержки 750 мс
mov BL, 4     ; частота повтора 20 символов/с
int 16h       ; вызываем прерывание
```

3.2.5. Функция *04h*

Эта функция позволяет управлять установкой звукового сигнала для нажатия клавиш. При активизации сигнала каждое нажатие будет сопровождаться характерным щелчком.

Использование:

1. В регистр AH следует поместить код функции 04h.
2. В регистр AL необходимо занести управляющий код (1 — включить звуковой сигнал, 0 — отключить звуковой сигнал).
3. Вызвать прерывание int 16h.

Функция не имеет специальных выходных параметров. Пример кода, использующий данную функцию, приведен в листинге 3.5.

Листинг 3.5. Использование функции 04h

```
mov AH, 4     ; код функции 04h
mov AL, 1     ; включить звуковой сигнал
int 16h       ; вызываем прерывание
```

3.2.6. Функция *05h*

Данная функция позволяет записать определенный символ в буфер клавиатуры. Особенность функции состоит в том, что она позволяет имитировать нажатия клавиш.

Использование:

1. В регистр AH следует поместить код функции 05h.
2. В регистр CH необходимо занести скан-код нужной клавиши.
3. В регистр CL нужно записать ASCII-код той же клавиши.
4. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр AL будет хранить результат операции: 00h — функция успешно завершена, 01h — функция не выполнена из-за переполнения буфера клавиатуры. Если функция не завершена, будет установлен флаг CF. Пример записи в буфер клавиатуры символа 'a' показан в листинге 3.6.

Листинг 3.6. Использование функции 05h

```
mov AH, 5    ; код функции 05h
mov CH, 30   ; скан-код буквы 'a' равен 1Eh
mov CL, 'a'  ; ASCII-код буквы 'a'
int 16h      ; вызываем прерывание
```

Кроме того, данную функцию можно использовать для записи в буфер какой-либо команды DOS (например, DATE), используя только ASCII-коды (листинг 3.7).

Листинг 3.7. Использование функции 05h для выполнения строковой команды

```
mov  CL, 'd'   ; запишем в регистр букву 'd'
call setbuffer ; и отправим в буфер клавиатуры
mov  CL, 'a'   ; запишем в регистр букву 'a'
call setbuffer ; и отправим в буфер клавиатуры
mov  CL, 't'   ; запишем в регистр букву 't'
call setbuffer ; и отправим в буфер клавиатуры
mov  CL, 'e'   ; запишем в регистр букву 'e'
call setbuffer ; и отправим в буфер клавиатуры
mov  CL, 0Ah   ; запишем в регистр символ перевода строки
call setbuffer ; и отправим в буфер клавиатуры
mov  CL, 0Dh   ; запишем в регистр символ возврата каретки
call setbuffer ; и тоже отправим в буфер клавиатуры
ret            ; выходим из программы
setbuffer proc ; процедура для записи байта в буфер клавиатуры
mov  AH, 5     ; код функции 05h
mov  CH, 0     ; скан-код не используем
int  16h       ; вызываем прерывание
setbuffer endp ; окончание процедуры
```

3.2.7. Функция *09h*

Эта функция позволяет определить набор функций, поддерживаемых клавиатурой, и, соответственно, узнать ее тип (84, 102 или 122 клавиши). Оп-

ределить, поддерживается ли эта функция BIOS, можно с помощью функции C0h прерывания int 15h.

Использование:

1. В регистр AH следует поместить код функции 09h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр AL будет хранить битовую маску поддерживаемых функций. Формат битовой маски представлен в табл. 3.8.

Таблица 3.8. *Битовая маска набора функций*

Бит	Описание
0	Поддержка функции 0300h прерывания int 16h (1 — поддерживается, 0 — не поддерживается)
1	Поддержка функции 0304h прерывания int 16h (1 — поддерживается, 0 — не поддерживается)
2	Поддержка функции 0305h прерывания int 16h (1 — поддерживается, 0 — не поддерживается)
3	Поддержка функции 0306h прерывания int 16h (1 — поддерживается, 0 — не поддерживается)
4	Поддержка функции 0Ah прерывания int 16h (1 — поддерживается, 0 — не поддерживается)
5	Поддержка функций 10h, 11h и 12h прерывания int 16h (1 — поддерживаются, 0 — не поддерживаются)
6	Поддержка функций 20h, 21h и 22h прерывания int 16h (1 — поддерживаются, 0 — не поддерживаются)
7	Резерв

Например, чтобы узнать, поддерживает ли клавиатура функции 20h—22h (122-клавишная), нужно написать код, показанный в листинге 3.8.

Листинг 3.8. Использование функции 09h

```
mov AH, 9     ; код функции 09h
int 16h       ; вызываем прерывание
and Al, 40h   ; проверяем бит 6
jnz ErrorHnd  ; функции не поддерживаются
```

3.2.8. Функция *0Ah*

Функция позволяет получить идентификатор установленной клавиатуры. Перед вызовом этой функции следует вызвать функцию 09h, чтобы убедиться в ее поддержке BIOS.

Использование:

1. В регистр AH следует поместить код функции 0Ah.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр BX будет хранить идентификатор клавиатуры: 0000h — клавиатура отсутствует, 41ABh, 54ABh, 83ABh, 84ABh — японская клавиатура, 86ABh — 122-клавишная клавиатура. Рассмотрим пример получения идентификатора подключенной клавиатуры (листинг 3.9).

Листинг 3.9. Использование функции 0Ah

```
mov AH, 0Ah    ; код функции 0Ah
int 16h        ; вызываем прерывание
cmp BX, 0      ; проверяем наличие клавиатуры
jz  NoKeyb     ; клавиатура не подключена
cmp BX, 86ABh ; 122 клавиши
jz  NoKeyb_122; не 122-клавишная
```

3.2.9. Функция *10h*

Данная функция позволяет прочитать любой символ с расширенной клавиатуры. Похожа на функцию 00h, но поддерживает расширенный набор символов.

Использование:

1. В регистр AH следует поместить код функции 10h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции в регистр AH будет помещен скан-код BIOS символа из расширенного набора, а в регистр AL — ASCII-код символа.

3.2.10. Функция *11h*

Функция позволяет проверить нажатие клавиши на расширенной клавиатуре. Если клавиша была нажата, функция запишет в соответствующие регистры скан-код и ASCII-код клавиши. Похожа на функцию 00h, только поддерживает расширенный набор символов.

Использование:

1. В регистр AH следует поместить код функции 11h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции в регистр AH будет помещен расширенный скан-код BIOS символа, а в регистр AL — ASCII-код символа. Кроме того, если нажатия клавиши не было, флаг ZF будет установлен в 1. Если клавиша была нажата, флаг ZF будет сброшен.

3.2.11. Функция *12h*

Эта функция позволяет получить состояние специальных клавиш на расширенной клавиатуре: <Shift>, <Ctrl>, <Alt>, <Num Lock>, <Scroll Lock>, <Caps Lock>, <SysReq> и <Insert>.

Использование:

1. В регистр AH следует поместить код функции 12h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр AX будет хранить значение состояния специальных клавиш. Формат возвращаемого значения представлен в табл. 3.9.

***Таблица 3.9.** Формат значения о состоянии расширенной клавиатуры*

Бит	Описание
0	Правая клавиша Shift (1 — нажата, 0 — не нажата)
1	Левая клавиша Shift (1 — нажата, 0 — не нажата)
2	Любая клавиша Ctrl (1 — нажата, 0 — не нажата)
3	Любая клавиша Alt (1 — нажата, 0 — не нажата)
4	Переключатель Scroll Lock (1 — включен, 0 — выключен)
5	Переключатель Num Lock (1 — включен, 0 — выключен)
6	Переключатель Caps Lock (1 — включен, 0 — выключен)
7	Переключатель Insert (1 — включен, 0 — выключен)
8	Левая клавиша Ctrl (1 — нажата, 0 — не нажата)
9	Левая клавиша Alt (1 — нажата, 0 — не нажата)
10	Правая клавиша Ctrl (1 — нажата, 0 — не нажата)
11	Правая клавиша Alt (1 — нажата, 0 — не нажата)

Таблица 3.9 (окончание)

Бит	Описание
12	Клавиша Scroll Lock (1 — нажата, 0 — не нажата)
13	Клавиша Num Lock (1 — нажата, 0 — не нажата)
14	Клавиша Caps Lock (1 — нажата, 0 — не нажата)
15	Клавиша SysReq (1 — нажата, 0 — не нажата)

3.2.12. Функция *20h*

Функция позволяет прочитать любой символ с расширенной клавиатуры. Похожа на функцию `10h`, только поддерживает 122-клавишные клавиатуры.

Использование:

1. В регистр `AH` следует поместить код функции `20h`.
2. Вызвать прерывание `int 16h`.

Выход:

После выполнения функции в регистр `AH` будет помещен скан-код BIOS символа из расширенного набора, а в регистр `AL` — ASCII-код символа.

3.2.13. Функция *21h*

Данная функция позволяет проверить нажатие клавиши на расширенной клавиатуре. Если клавиша была нажата, функция запишет в соответствующие регистры скан-код и ASCII-код клавиши. Похожа на функцию `11h`, только поддерживает 122-клавишные клавиатуры.

Использование:

1. В регистр `AH` следует поместить код функции `21h`.
2. Вызвать прерывание `int 16h`.

Выход:

После выполнения функции в регистр `AH` будет помещен расширенный скан-код BIOS символа, а в регистр `AL` — ASCII-код символа. Кроме того, если нажатия клавиши не было, флаг `ZF` будет установлен в 1. Если клавиша была нажата, флаг `ZF` будет сброшен.

3.2.14. Функция *22h*

Эта функция позволяет получить состояние специальных клавиш на расширенной клавиатуре: <Shift>, <Ctrl>, <Alt>, <Num Lock>, <Scroll Lock>,

<Caps Lock>, <SysReq> и <Insert>. Похожа на функцию 11h, только поддерживает 122-клавишные клавиатуры.

Использование:

1. В регистр AH следует поместить код функции 22h.
2. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр AX будет хранить значение состояния специальных клавиш. Формат возвращаемого значения был представлен в табл. 3.9.

3.2.15. Функция *FFh*

Функция позволяет добавить символ в конец буфера клавиатуры.

Использование:

1. В регистр AH следует поместить код функции FFh.
2. В регистр DX следует записать скан-код желаемого символа.
3. Вызвать прерывание int 16h.

Выход:

После выполнения функции регистр AL будет хранить результат операции: 00h — успешное выполнение, 01h — произошла ошибка. Попробуем в следующем примере (листинг 3.10) записать в буфер клавиатуры скан-код буквы 'Q'.

Листинг 3.10. Использование функции FFh

```
mov  AH, 0FFh ; код функции FFh
mov  DX, 10h  ; скан-код буквы 'Q' равен 10h
int  16h      ; вызываем прерывание
test AL, AL   ; проверка на 0
jz   ErrorHnd ; произошла ошибка
```

На этом примере можно завершить рассмотрение функций BIOS и перейти к программированию портов контроллера клавиатуры.

3.3. Использование портов

Клавиатура и мышь в современных компьютерах работают по протоколу PS/2 и обе подключаются к общему контроллеру клавиатуры (например, Intel 8042), интегрированному в чип (микросхему) на материнской плате.

Для клавиатуры контроллер генерирует прерывание IRQ1. Контроллер может работать в двух режимах: PS/2-совместимом и AT-совместимом. Первый режим предусматривает поддержку двух устройств: мыши и клавиатуры. Доступ к контроллеру клавиатуры осуществляется через порты 60h и 64h. Порт 60h в режиме чтения получает данные из буфера клавиатуры, а в режиме записи помещает данные в буфер. Порт 64h тоже работает в двух режимах: в режиме чтения он является регистром состояния, а в режиме записи выполняет роль регистра команд. Формат регистра состояния (64h) представлен в табл. 3.10.

Таблица 3.10. *Регистр состояния (64h)*

Бит	Описание
0	Наличие данных в выходном буфере клавиатуры (0 — выходной буфер пустой, 1 — в буфере есть данные)
1	Наличие данных во входном буфере клавиатуры (0 — входной буфер пустой, 1 — в буфере есть данные)
2	Результат самотестирования (0 — сброс, 1 — тест прошел успешно)
3	Порт, используемый для последней операции (0 — 60h, 1 — 64h)
4	Состояние клавиатуры (0 — заблокирована, 1 — включена)
5	Ошибка передачи (0 — ошибок нет, 1 — клавиатура не отвечает)
6	Ошибка тайм-аута (0 — ошибка отсутствует, 1 — ошибка)
7	Ошибка четности (0 — ошибка отсутствует, 1 — ошибка) указывает на последнюю ошибку, произошедшую при передаче данных

Формат регистра команд (64h) показан в табл. 3.11.

Таблица 3.11. *Регистр команд (64h)*

Бит	Описание
0	Прерывания для клавиатуры (0 — отключить, 1 — включить)
1	Прерывания для мыши (0 — отключить, 1 — включить)
2	Системный флаг (1 — инициализация через самотестирование, 0 — инициализация по питанию)
3	Не используется
4	Доступ к клавиатуре (0 — открыт, 1 — закрыт)
5	Доступ к мыши (0 — открыт, 1 — закрыт)
6	Трансляция скан-кодов (0 — не использовать, 1 — использовать)
7	Резерв

Перед началом работы с клавиатурой следует проверить наличие данных в буфере (бит 0 в регистре статуса). Кроме того, такую проверку необходимо выполнять перед любыми последующими операциями записи. После проверки буфера в регистр 64h записывается код желаемой команды (табл. 3.12). Если команда имеет дополнительные параметры, их следует записать в регистр данных (60h). После выполнения команды в регистр данных (60h) будет помещен результат (табл. 3.13).

Таблица 3.12. *Команды управления контроллером клавиатуры*

Код команды	Описание
20h	Прочитать байт из регистра команд
60h	Записать байт в регистр команд
A1h	Получить номер версии производителя
A4h	Получить пароль (возвратит FAh, если пароль существует, и F1h — в обратном случае)
A5h	Установить пароль (посылает строку с нулевым символом в конце)
A6h	Проверить пароль (сравнивает введенный с клавиатуры пароль с текущим)
AAh	Выполнить самотестирование контроллера (в случае успеха возвратит 55h)
ABh	Проверка интерфейса клавиатуры (00h — все хорошо, 01h — низкий уровень сигнала синхронизации, 02h — высокий уровень сигнала синхронизации, 03h — низкий уровень сигнала на линии данных, 04h — высокий уровень сигнала на линии данных)
ADh	Отключить интерфейс клавиатуры (устанавливает бит 4 в регистре команд)
AEh	Включить интерфейс клавиатуры (очищает бит 4 в регистре команд)
AFh	Получить версию
C0h	Прочитать входной порт
D0h	Прочитать значение из выходного порта
D1h	Записать параметр в выходной порт
D2h	Записать параметр в буфер клавиатуры
E0h	Тестирование порта (возвращает тестовое значение для порта)

Рассмотрим несколько примеров использования управляющих команд. Сначала напишем код, который выполняет проверку интерфейса (команда ABh) клавиатуры (листинг 3.11).

Таблица 3.13. *Коды ошибок клавиатуры*

Код ошибки	Описание
00h	Ошибка переполнения (слишком много нажатий клавиш)
AAh	Самотестирование контроллера успешно завершено
EEh	Результат эхо-режима, инициированного командой EEh
FAh	Подтверждение успешного выполнения команды
FCh	Ошибка самотестирования клавиатуры
FEh	Запрос на повторную передачу данных (команда Resend)
FFh	Ошибка клавиатуры

Листинг 3.11. Проверка наличия клавиатуры

```
; ждем освобождения входного буфера клавиатуры
@keyb_wait:
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 010b  ; если буфер занят
jnz  @keyb_wait; повторяем опрос
mov  AL, 0ABh  ; команду тестирования интерфейса
out  64h, AL   ; записываем в порт
```

Для программистов C++ этот пример представлен в листинге 3.12.

Листинг 3.12. Проверка наличия клавиатуры в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( ( dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду проверки интерфейса
outPort ( 0x64, 0xAB, 1);
```

Теперь попробуем отключить клавиатуру. Для этого применим управляющую команду с кодом ADh так, как это сделано в листинге 3.13.

Листинг 3.13. Отключение клавиатуры

```
; ждем освобождения входного буфера клавиатуры
@keyb_wait:
in   AL, 64h    ; опрашиваем регистр состояния
test AL, 010b   ; если буфер занят
jnz  @keyb_wait; повторяем опрос
mov  AL, 0ADh   ; команду отключения клавиатуры
out  64h, AL    ; записываем в порт
```

Аналогичный пример для C++ показан в листинге 3.14.

Листинг 3.14. Отключение клавиатуры в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x64, &dwResult, 1);
      if ( (dwResult & 0x02) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем в порт команду отключения клавиатуры
outPort ( 0x64, 0xAD, 1);
```

Чтобы восстановить работу клавиатуры, запишите в регистр команд код AEh. Как видно из примеров, работа с командами управления не вызывает проблем. Кроме перечисленных ранее команд управления контроллером клавиатуры, существует еще ряд дополнительных команд, которые применяются для настройки различных режимов работы устройства и управления встроенным в клавиатуру процессором. Список этих команд представлен в табл. 3.14. Команды могут иметь дополнительные параметры, которые передаются в виде дополнительного байта. Код команды следует записать в порт 60h. После посылки каждой команды следует проверить бит 1 для порта 64h и только при его сбросе (равенстве 0) передавать дополнительный параметр. Кроме того, можно проверить нулевой бит в порту 60h. После выполнения каждой команды, кроме EEh и FEh, возвращается подтверждающий код FAh (читается из порта 60h). Если значение кода команды или параметра недо-

пустимы, клавиатура вернет код FEh. Если вместо дополнительного байта параметра послать код новой команды, клавиатура выполнит именно его, "забыв" о предыдущей команде.

Таблица 3.14. *Набор команд для клавиатуры*

Код команды	Описание
EDh	Установить или сбросить состояние индикаторов
EEh	Выполнить эхо-диагностику клавиатуры
F0h	Выбрать набор скан-кодов (первый, второй или третий)
F2h	Получить идентификатор клавиатуры
F3h	Настроить параметры автоповтора и время задержки
F4h	Включить клавиатуру (буфер будет очищен)
F5h	Отключить клавиатуру и загрузить значения по умолчанию
F6h	Установить значения параметров, используемых по умолчанию (10 символов/с, 500 мс)
F7h	Установить режим автоповтора для всех клавиш
F8h	Установить для всех клавиш передачу кодов нажатия и отпускания
F9h	Установить для всех клавиш передачу кодов нажатия
FAh	Установить для всех клавиш передачу кодов нажатия, отпускания и режим автоповтора
FBh	Установить для указанной клавиши передачу кодов нажатия и режим автоповтора
FCh	Установить для указанной клавиши передачу кодов нажатия и отпускания
FDh	Установить для указанной клавиши передачу кодов нажатия
FEh	Выполнить повторную передачу последнего переданного байта
FFh	Сброс клавиатуры

Некоторые команды требуют дополнительного описания, поэтому рассмотрим их более подробно.

3.3.1. Команда *EDh*

Данная команда позволяет управлять состоянием индикаторов на клавиатуре. Размер команды равен двум байтам. Первый байт содержит сам код команды, а второй — битовую маску для настройки индикаторов (табл. 3.15).

Таблица 3.15. *Байт состояния индикаторов*

Бит	Описание
0	Индикатор Scroll Lock (1 — включен, 0 — выключен)
1	Индикатор Num Lock (1 — включен, 0 — выключен)
2	Индикатор Caps Lock (1 — включен, 0 — выключен)
3–7	Резерв

Приведу простой пример для управления индикатором <Num Lock> (листинг 3.15).

Листинг 3.15. Управление индикатором <Num Lock>

```
@keyb_wait:
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 010b  ; если буфер занят
jnz  @keyb_wait; повторяем опрос
mov  AL, 0EDh  ; команду управления индикаторами
out  60h, AL   ; записываем в порт данных
@data_wait:
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 010b  ; если буфер занят
jnz  @data_wait; повторяем опрос
mov  AL, 010b  ; включаем Num Lock
out  60h, AL   ; записываем в порт значение
```

Для C++ аналогичный пример показан в листинге 3.16.

Листинг 3.16. Управление индикатором <Num Lock> в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
```

```
// записываем в порт команду управления индикаторами
outPort ( 0x60, 0xED, 1);
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
{
       // читаем состояние порта
       inPort ( 0x64, &dwResult, 1);
       if ( (dwResult & 0x02) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return MY_ERROR_TIME;
}
outPort ( 0x60, 0x02, 1);
```

3.3.2. Команда *EEh*

Команда позволяет протестировать клавиатуру на предмет работоспособности. Если в работе клавиатуры возникли сбои, следует сделать сброс (команда FFh) и послать эту команду. Возвращаемое значение, отличное от EEh, явно укажет на сбои в работе клавиатуры. Возможно, придется даже отключить и повторно включить питание для восстановления нормальной работоспособности устройства.

3.3.3. Команда *F2h*

Эта команда позволяет получить идентификатор клавиатуры и убедиться в ее наличии. После выполнения команды клавиатура вернет код подтверждения FAh, а затем идентификатор: для большинства клавиатур это два кода — ABh и 83h (для отдельных клавиатур 41h). Рассмотрим простой пример получения идентификатора клавиатуры, описанный в листинге 3.17.

Листинг 3.17. Получение идентификатора клавиатуры

```
@keyb_wait:
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 010b  ; если буфер занят
jnz  @keyb_wait; повторяем опрос
mov  AL, 0F2h  ; команду эхо-диагностики
out  60h, AL   ; записываем в порт данных
call @data_wait
in   AL, 60h   ; получаем подтверждение
cmp  AL, 0FAh  ; выполнения команды
jne  ErrorHnd  ; произошла ошибка
```

```
call @data_wait
in   AL, 60h    ; получаем подтверждение
cmp  AL, 0ABh   ; первый байт идентификатора
jne  ErrorHnd   ; произошла ошибка
call @data_wait
in   AL, 60h    ; получаем подтверждение
cmp  AL, 83h    ; второй байт идентификатора
jne  ErrorHnd   ; произошла ошибка
ret
@data_wait:
in   AL, 64h    ; опрашиваем регистр состояния
test AL, 01b    ; если буфер занят
jnz  @data_wait; повторяем опрос
ret
```

В данном примере мы послали команду `F2h` контроллеру клавиатуры и последовательно получили три байта: байт подтверждения и два байта идентификатора клавиатуры. Аналогичный код для C++ приведен в листинге 3.18.

Листинг 3.18. Получение идентификатора клавиатуры в C++

```
// для удобства работы пишем отдельную функцию опроса данных в буфере
bool WaitData ()
{
      // переменная для хранения результата
      DWORD dwResult  = 0;
      int iTimeWait = 50000;
      // проверяем наличие данных во входном буфере клавиатуры
      while ( -- iTimeWait > 0)
      {
          // читаем состояние порта
          inPort ( 0x64, &dwResult, 1);
          if ( (dwResult & 0x01) == 0x00) return true;
          // закончилось время ожидания
          if ( iTimeWait < 1) return false;
      }
return false;
}
// основной код программы
DWORD dwResult  = 0;
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
```

```
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x02) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
outPort ( 0x60, 0xF2, 1); // записываем код команды
// ждем получения первого байта
iTimeWait = 50000;
if ( WaitData ()) // получен
{
    inPort ( 0x60, &dwResult, 1);
    // проверяем на равенство коду FAh
    if ( dwResult != 0xFA)
      return MY_ERROR;
}
else // нет данных
    return MY_ERROR;
// ждем получения второго байта
iTimeWait = 50000;
if ( WaitData ()) // получен
{
    inPort ( 0x60, &dwResult, 1);
    // проверяем на равенство коду ABh
    if ( dwResult != 0xAB)
      return MY_ERROR;
}
else // нет данных
    return MY_ERROR;
// ждем получения третьего байта
iTimeWait = 50000;
if ( WaitData ()) // получен
{
    inPort ( 0x60, &dwResult, 1);
    // проверяем на равенство коду 83h
    if ( dwResult != 0x83)
      return MY_ERROR;
}
else // нет данных
    return MY_ERROR;
// идентификатор клавиатуры успешно получен
```

3.3.4. Команда *F3h*

Данная команда позволяет установить частоту и время задержки для автоматического повтора набранного на клавиатуре символа. Размер команды равен двум байтам. Первый байт содержит сам код команды, а второй — описание параметров настройки автоповтора (табл. 3.16).

Таблица 3.16. *Содержание байта настройки автоповтора*

Биты	Описание
0—4	Код значения частоты автоповтора
5—6	Код значения времени задержки
7	Резерв

Коды значений частоты автоповтора и времени задержки были представлены соответственно в табл. 3.7 и 3.6. Простой пример использования команды `F3h` показан в листинге 3.19.

Листинг 3.19. Использование команды `F3h`

```
@keyb_wait:
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 010b  ; если буфер занят
jnz  @keyb_wait; повторяем опрос
mov  AL, 0F3h  ; команду F3h
out  60h, AL   ; записываем в порт данных
@data_wait:    ; есть ли ответ
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 01b   ; на наличие данных
jz   @data_wait; повторяем опрос
in   AL, 60h   ; получаем подтверждение
cmp  AL, 0FAh  ; выполнения команды
jne  ErrorHnd  ; произошла ошибка
@keyb_wait:
in   AL, 64h   ; опрашиваем регистр состояния
test AL, 010b  ; если буфер занят
jnz  @keyb_wait; повторяем опрос
mov  AL, 28h   ; 15 символов/с и 750 мс
out  60h, AL   ; записываем в порт данных
```

Аналогичный код для установки параметров автоповтора (частота 15 символов/с и время 750 мс) для C++ представлен в листинге 3.20.

Листинг 3.20. Использование команды F3h в C++

```
DWORD dwResult  = 0;
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x02) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
outPort ( 0x60, 0xF3, 1); // записываем код команды
int iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
 {
    // читаем состояние порта
    inPort ( 0x64, &dwResult, 1);
    if ( (dwResult & 0x01) == 0x00) break;
    // закончилось время ожидания
    if ( iTimeWait < 1) return MY_ERROR_TIME;
 }
inPort ( 0x60, &dwResult, 1);
// проверяем на равенство коду FAh
    if ( dwResult != 0xFA) return MY_ERROR;
iTimeWait = 50000;
// проверяем наличие данных во входном буфере клавиатуры
while ( -- iTimeWait > 0)
{
        // читаем состояние порта
        inPort ( 0x64, &dwResult, 1);
        if ( (dwResult & 0x02) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return MY_ERROR_TIME;
}
// записываем значение частоты и время задержки
outPort ( 0x60, 0x28, 1);
```

Остальные функции управления достаточно просты и не требуют дополнительных примеров. Если сравнить программирование клавиатуры с использованием прерывания `int 16h` и непосредственно портов, то первый способ

окажется гораздо более удобным. Однако доступ посредством аппаратных портов дает выигрыш (незначительный) в скорости и позволяет применять весь набор команд для любых случаев. В конечном итоге выбор остается за программистом. Для удобства работы приведу дополнительно скан-коды клавиш (второй набор), используемых во всех современных клавиатурах (табл. 3.17).

Таблица 3.17. *Второй набор скан-кодов клавиатуры*

Клавиша	Код нажатия	Код отпускания	Клавиша	Код нажатия	Код отпускания
F1	05	F0, 05	‘	0E	F0, 0E
F2	06	F0, 06	-	4E	F0, 4E
F3	04	F0, 04	=	55	F0, 55
F4	0C	F0, 0C	Backspace	66	F0, 66
F5	03	F0, 03	Tab	0D	F0, 0D
F6	0B	F0, 0B	Space	29	F0, 29
F7	83	F0, 83	Caps Lock	58	F0, 58
F8	0A	F0, 0A	Esc	76	F0, 76
F9	01	F0, 01	Enter	5A	F0, 5A
F10	09	F0, 09	Left Ctrl	14	F0, 14
F11	78	F0, 78	Right Ctrl	E0, 14	E0, F0, 14
F12	07	F0, 07	Left Alt	11	F0, 11
0	45	F0, 45	Right Alt	E0, 11	E0, F0, 11
1	16	F0, 16	Left Shift	12	F0, 12
2	1E	F0, 1E	Right Shift	59	F0, 59
3	26	F0, 26	Left Win	E0, 1F	E0, F0, 1F
4	25	F0, 25	Right Win	E0, 27	E0, F0, 27
5	2E	F0, 2E	Apps	E0, 2F	E0, F0, 2F
6	36	F0, 36	Print Screen	E0, 12, E0 7C	E0, F0, 7C E0, F0, 12
7	3D	F0, 3D	Scroll Lock	7E	F0, 7E
8	3E	F0, 3E	Pause	E1, 14, 77 E1, F0, 14 F0, 77	Нет
9	46	F0, 46	[	54	F0, 54

Таблица 3.17 (продолжение)

Клавиша	Код нажатия	Код отпускания	Клавиша	Код нажатия	Код отпускания
A	1C	F0, 1C	]	5B	F0, 5B
B	32	F0, 32	;	4C	F0, 4C
C	21	F0, 21	|	52	F0, 52
D	23	F0, 23	,	41	F0, 41
E	24	F0, 24	.	49	F0, 49
F	2B	F0, 2B	/	4A	F0, 4A
G	34	F0, 34	Insert	E0, 70	E0, F0, 70
H	33	F0, 33	Home	E0, 6C	E0, F0, 6C
I	43	F0, 43	Delete	E0, 71	E0, F0, 71
J	3B	F0, 3B	End	E0, 69	E0, F0, 69
K	42	F0, 42	Page Up	E0, 7D	E0, F0, 7D
L	4B	F0, 4B	Page Down	E0, 7A	E0, F0, 7A
M	3A	F0, 3A	Up Arrow	E0, 75	E0, F0, 75
N	31	F0, 31	Down Arrow	E0, 72	E0, F0, 72
O	44	F0, 44	Left Arrow	E0, 6B	E0, F0, 6B
P	4D	F0, 4D	Right Arrow	E0, 74	E0, F0, 74
Q	15	F0, 15	Num Lock	77	F0, 77
R	2D	F0, 2D	(+) Enter	E0, 5A	E0, F0, 5A
S	1B	F0, 1B	(+) /	E0, 4A	E0, F0, 4A
T	2C	F0, 2C	(+) *	7C	F0, 7C
U	3C	F0, 3C	(+) -	7B	F0, 7B
V	2A	F0, 2A	(+) +	79	F0, 79
W	1D	F0, 1D	(+) .	71	F0, 71
X	22	F0, 22	(+) 0	70	F0, 70
Y	35	F0, 35	(+) 1	69	F0, 69
Z	1A	F0, 1A	(+) 2	72	F0, 72
(IE) Search	E0, 10	E0, F0, 10	(+) 3	7A	F0, 7A
(IE) Home	E0, 3A	E0, F0, 3A	(+) 4	6B	F0, 6B
(IE) Stop	E0, 28	E0, F0, 28	(+) 5	73	F0, 73

Таблица 3.17 (окончание)

Клавиша	Код нажатия	Код отпускания	Клавиша	Код нажатия	Код отпускания
(IE) Refresh	E0, 20	E0, F0, 20	(+) 6	74	F0, 74
(IE) Forward	E0, 30	E0, F0, 30	(+) 7	6C	F0, 6C
(IE) Back	E0, 38	E0, F0, 38	(+) 8	75	F0, 75
(IE) Favorites	E0, 18	E0, F0, 18	(+) 9	7D	F0, 7D
Volume Up	E0, 32	E0, F0, 32	Next Track	E0, 4D	E0, F0, 4D
Volume Down	E0, 21	E0, F0, 21	Previous Track	E0, 15	E0, F0, 15
Mute	E0, 23	E0, F0, 23	Wake	E0, 5E	E0, F0, 5E
Play	E0, 34	E0, F0, 34	Power	E0, 37	E0, F0, 37
Stop	E0, 3B	E0, F0, 3B	Sleep	E0, 3F	E0, F0, 3F

Существует еще один порт `61h`, используемый в современных компьютерах для управления встроенным динамиком (спикером). Он служит для чтения и записи и позволяет управлять некоторыми функциями клавиатуры (ради совместимости со старыми моделями). Формат регистра `61h` представлен в табл. 3.18.

Таблица 3.18. Формат регистра *61h*

Бит	Описание
0–5	Для клавиатуры не используются
6	Низкий уровень сигнала на линии синхронизации
7	Отключение клавиатуры (1 — выключить, 0 — включить)

Как видно из таблицы, для работы с клавиатурой отведены только два бита: 6 и 7. Остальные изменять не рекомендуется. Например, чтобы отключить клавиатуру, следует написать код, приведенный в листинге 3.21.

Листинг 3.21. Отключение клавиатуры с помощью портов

```
in  AL, 61h    ; прочитать состояние порта
or  AL, 80h    ; выполнить побитовую операцию ИЛИ
out 61h, AL    ; выключить клавиатуру
```

Для включения заблокированной клавиатуры можно использовать код из листинга 3.22.

Листинг 3.22. Включение клавиатуры с помощью портов

```
in  AL, 61h        ; прочитать состояние порта
and AL, 01111111h ; нужно обнулить бит 7 для включения клавиатуры
out 61h, AL        ; записать значение в порт
```

Для C++ аналогичный код представлен в листинге 3.23.

Листинг 3.23. Включение и отключение клавиатуры с помощью портов в C++

```
// пишем функцию управления клавиатурой
void LockUnlockKeyboard ( bool bLock)
{
   DWORD dwResult = 0;
   if ( bLock) // блокировать
   {
     inPort ( 0x61, dwResult, 1);
     dwResult |= 0x80;
     outPort ( 0x61, dwResult, 1);
   }
   else // разблокировать
   {
     inPort ( 0x61, dwResult, 1);
     dwResult &= 0x7F;
     outPort ( 0x61, dwResult, 1);
   }
}
```

А теперь посмотрим, какие возможности по управлению клавиатурой предоставляет программный интерфейс Win32.

3.4. Использование Win32 API

Возможности интерфейса Win32 API "традиционно" скромны и ограниченны. Мы условно разделим описание всех возможностей на несколько частей:

1. Настройка клавиатуры.
2. Использование "горячих" клавиш и акселераторов.
3. Поддержка различных языков.

Поддержка клавиатуры в операционных системах Windows осуществляется независимо от типа подключенного устройства и используемого языка. Любые нажатия клавиш (в виде скан-кодов) преобразуются системой (драйвером клавиатуры) и, в зависимости от установленного языка, отображаются

в окне запущенного приложения. Каждому языку соответствует свой набор символов, а драйвер сам выполняет все необходимые преобразования. Программисту достаточно ознакомиться только с универсальным набором скан-кодов так называемых виртуальных клавиш (основные из них перечислены в табл. 3.19). Каждая виртуальная клавиша имеет свое имя и числовое значение. В программах следует пользоваться по возможности именами клавиш, чтобы сохранить совместимость программ в дальнейшем.

Таблица 3.19. *Виртуальные клавиши*

Клавиша	**Имя**	**Клавиша**	**Имя**
Tab	VK_TAB	Print Screen	VK_SNAPSHOT
Backspace	VK_BACK	Scroll Lock	VK_SCROLL
Enter	VK_RETURN	Pause	VK_PAUSE
Shift	VK_SHIFT	Insert	VK_INSERT
Alt	VK_MENU	Delete	VK_DELETE
Ctrl	VK_CONTROL	Home	VK_HOME
Caps Lock	VK_CAPITAL	End	VK_END
Esc	VK_ESCAPE	Page Up	VK_PRIOR
Space	VK_SPACE	Page Down	VK_NEXT
Left Arrow	VK_LEFT	Left Win	VK_LWIN
Right Arrow	VK_RIGHT	Right Win	VK_RWIN
Up Arrow	VK_UP	Apps	VK_APPS
Down Arrow	VK_DOWN	Sleep	VK_SLEEP
(+) 0	VK_NUMPAD0	*	VK_MULTIPLY
(+) 1	VK_NUMPAD1	+	VK_ADD
(+) 2	VK_NUMPAD2	\	VK_SEPARATOR
(+) 3	VK_NUMPAD3	-	VK_SUBTRACT
(+) 4	VK_NUMPAD4	/ (деление)	VK_DIVIDE
(+) 5	VK_NUMPAD5	Num Lock	VK_NUMLOCK
(+) 6	VK_NUMPAD6	Left Shift	VK_LSHIFT
(+) 7	VK_NUMPAD7	Right Shift	VK_RSHIFT
(+) 8	VK_NUMPAD8	Left Ctrl	VK_LCONTROL
(+) 9	VK_NUMPAD9	Right Ctrl	VK_RCONTROL

Таблица 3.19 (окончание)

Клавиша	Имя	Клавиша	Имя
Left Alt	VK_LMENU	F6	VK_F6
Right Alt	VK_LMENU	F7	VK_F7
F1	VK_F1	F8	VK_F8
F2	VK_F2	F9	VK_F9
F3	VK_F3	F10	VK_F10
F4	VK_F4	F11	VK_F11
F5	VK_F5	F12	VK_F12

Для буквенно-цифровых клавиш существуют только числовые коды, с которыми можно ознакомиться в документации фирмы Microsoft.

3.4.1. Настройка клавиатуры

Итак, первая возможность настройки клавиатуры, которую мы рассмотрим, относится к установке времени задержки для автоповтора. Для этого используется уже знакомая нам по *гл. 2* функция SystemParametersInfo с параметром SPI_SETKEYBOARDDELAY. Второй аргумент функции должен указывать на время задержки. Поддерживаются четыре значения времени: 0 — 250 мс, 1 — 500 мс, 2 — 750 мс, 3 — 1000 мс. Существует также возможность получения текущего значения времени задержки, для чего служит параметр SPI_GETKEYBOARDDELAY. Пример кода, использующий обе эти возможности, представлен в листинге 3.24.

Листинг 3.24. Установка и получение времени задержки

```
// напишем функцию для установки времени задержки
bool SetDelay ( int iDelay)
{
    int iOldDelay = -1;
    // проверяем диапазон аргумента (должен быть от 0 до 3)
    if ( ( iDelay < 0) && ( iDelay > 3)) return false;
    // получаем текущее значение времени задержки
    SystemParametersInfo ( SPI_GETKEYBOARDDELAY, 0, &iOldDelay, 0);
    // если значение ошибочно или равно устанавливаемому, выходим
    if ( ( iOldDelay == -1) || ( iOldDelay == iDelay)
      return false;
```

```
    // устанавливаем новое значение времени задержки
    SystemParametersInfo ( SPI_SETKEYBOARDDELAY, iDelay, 0,
        SPIF_SENDCHANGE | SPIF_UPDATEINIFILE);
return true;
}
```

Кроме времени задержки, существует также возможность изменения частоты автоповтора символов. Для этого следует передать в первый аргумент функции `SystemParametersInfo` параметр `SPI_SETKEYBOARDSPEED`, а во второй — новое значение частоты (от 0 до 31). Установка частоты в 0 соответствует 2,5 символам в секунду, а установка числа 31 — 30 символам в секунду. Остальные возможные значения лежат между ними (подробнее см. в табл. 3.7). Для получения текущего значения частоты автоповтора используется параметр `SPI_GETKEYBOARDSPEED`. В листинге 3.25 представлен пример функции, устанавливающей новое значение частоты автоповтора.

Листинг 3.25. Установка и получение частоты автоповтора

```
// напишем функцию для установки частоты автоповтора
bool SetFreq ( int iFreq)
{
    int iOldFreq = -1;
    // проверяем диапазон аргумента (должен быть от 0 до 31)
    if ( ( iFreq < 0) && ( iFreq > 31)) return false;
    // получаем текущее значение частоты
    SystemParametersInfo ( SPI_GETKEYBOARDSPEED, 0, &iOldFreq, 0);
    // если значение ошибочно или равно устанавливаемому, выходим
    if ( ( iOldFreq == -1) || ( iOldFreq == iFreq)
      return false;
    // устанавливаем новое значение частоты
    SystemParametersInfo ( SPI_SETKEYBOARDSPEED, iFreq, 0,
        SPIF_SENDCHANGE | SPIF_UPDATEINIFILE);
return true;
}
```

Как видите, настройка параметров автоповтора не представляет особых трудностей. Давайте теперь перейдем к еще одной удобной возможности управления клавиатурой. Она не имеет прямого отношения к устройству клавиатуры, но тесно связана с ней — это частота мигания текстового курсора. Для установки частоты мигания применяется функция `SetCaretBlinkTime`. Единственный аргумент функции задает промежуток времени (в миллисекундах) между двумя появлениями текстового курсора на экране. Существует также функция `GetCaretBlinkTime`, позволяющая получить текущее значение времени мигания курсора. Следует иметь в виду, что измене-

ние частоты мигания курсора будет использоваться системой для всех приложений. В листинге 3.26 представлен пример функции, устанавливающей новое значение для частоты мигания курсора.

Листинг 3.26. Установка частоты мигания текстового курсора

```
// напишем функцию для установки частоты мигания курсора
bool SetFreqCaret ( int iFreq)
{
    int iOldFreq = 200;
    // проверяем диапазон аргумента (должен быть от 200 до 1200)
    if ( ( iFreq < 200) && ( iFreq > 1200)) return false;
    // получаем текущее значение частоты
    iOldFreq = GetCaretBlinkTime ();
    // если значение равно устанавливаемому, выходим
    if ( iOldFreq == iFreq)
      return false;
    // устанавливаем новое значение частоты
    if ( !SetCaretBlinkTime ( iFreq))
      return false; // произошла ошибка
return true;
}
```

При успешном завершении функция `SetCaretBlinkTime` должна возвратить ненулевое значение. Если функция вернула 0, значит, произошла ошибка, расширенное описание которой можно получить через вызов `GetLastError`. То же самое относится и к функции `GetCaretBlinkTime`.

Существует возможность программного отключения клавиатуры. Для этого применяется функция `BlockInput`, подробно рассмотренная в *гл. 2*.

3.4.2. Использование "горячих" клавиш

Очень удобной возможностью работы в Windows является использование оперативных и "горячих" клавиш. Под оперативными клавишами понимают комбинации клавиш для быстрого доступа к различным командам меню. Их еще называют акселераторными клавишами. Удобство их заключается в том, что к ним можно обращаться, когда меню закрыто и не активно. Работать с акселераторами достаточно просто. Вначале, используя редактор ресурсов, следует добавить в свою программу ресурс акселератора, присвоить ему идентификатор (например, `MY_ACCEL`) и назначить комбинации клавиш. Далее следует открыть в этом же редакторе ресурс меню и добавить к желаемым пунктам описание закрепленной за ним клавиши (например, "`&Открыть \t Ctrl+O`"). После этого добавляем в стандартный код окна описание акселераторов, как показано в листинге 3.27.

Листинг 3.27. Добавление поддержки акселераторов в программу

```
// в функцию WinMain добавляем следующий код
// ...
// загружаем таблицу акселераторов с идентификатором MY_ACCEL
HACCEL hAccel = LoadAccelerators ( hInst, MAKEINTRESOURCE ( MY_ACCEL));
// ...
// добавляем в цикл обработки сообщений перехватчик акселераторных клавиш
while ( GetMessage ( &msg, NULL, 0, 0))
{
    if ( !TranslateAccelerator ( hWnd, hAccel, &msg))
      {
        TranslateMessage ( &msg);
        DispatchMessage ( &msg);
      }
}
```

Как видно из кода, для загрузки таблицы акселераторов используется функция `LoadAccelerators`. Данная функция имеет два аргумента. Первый аргумент указывает на дескриптор приложения, а второй определяет имя ресурса для таблицы акселераторов. Можно вместо имени использовать идентификатор, обработав его макросом `MAKEINTRESOURCE`. После загрузки таблицы акселераторов необходимо установить перехватчик для всех оперативных клавиш. Роль перехватчика выполняет функция `TranslateAccelerator`. Первый аргумент функции указывает на дескриптор главного окна программы. Второй аргумент должен хранить дескриптор таблицы акселераторов (тип `HACCEL`). Третий аргумент указывает на структуру `MSG`. Вот и все сложности. Теперь выбор назначенной для пункта меню комбинации клавиш (например, <Ctrl>+<O>) вызовет выполнение соответствующей команды.

К сожалению, у акселераторов есть один существенный недостаток. Вы не сможете их использовать, если приложение потеряло фокус. Операционная система Windows автоматически выбирает таблицу акселераторов только для активного окна. Решить эту проблему можно с помощью "горячих" клавиш. Они представляют собой такую же комбинацию клавиш, как и акселераторы, но имеют в системе глобальный статус. Независимо от того, какая программа активна в данный момент, можно вызывать различные команды посредством "горячих" клавиш. Согласитесь, это очень удобно: ваша программа может "сидеть" в системном трее или вообще быть невидимой, но стоит на клавиатуре набрать заветную комбинацию, и любая назначенная вами команда будет исполнена. Прежде чем рассказать, как программируются "горячие" клавиши, хочу выделить два важных правила:

1. Каждая "горячая" клавиша регистрируется в системе отдельно и должна иметь уникальный глобальный идентификатор.

2. После завершения работы с программой необходимо удалить зарегистрированные "горячие" клавиши из системы.

Для регистрации "горячей" клавиши используется функция RegisterHotKey. Функция имеет четыре аргумента. Первый указывает на дескриптор окна, которое будет обрабатывать специальное сообщение WM_HOTKEY. Второй аргумент указывает на идентификатор "горячей" клавиши, который назначается разработчиком. Это значение должно лежать в определенных пределах: для обычной программы — от 0x0000 до 0xBFFF, для библиотеки DLL — от 0xC000 до 0xFFFF. Чтобы избежать конфликтов с другими аналогичными значениями, для библиотек DLL следует использовать функцию GlobalAddAtom. Третий аргумент функции определяет модификатор для комбинации клавиш. Возможные значения для этого аргумента приведены в табл. 3.20. Можно комбинировать модификаторы между собой.

Таблица 3.20. *Типы модификатора клавиш*

Константа	Описание
MOD_SHIFT	Необходимо удерживать любую клавишу <Shift> на клавиатуре
MOD_CONTROL	Необходимо удерживать любую клавишу <Ctrl> на клавиатуре
MOD_ALT	Необходимо удерживать любую клавишу <Alt> на клавиатуре
MOD_WIN	Необходимо удерживать любую клавишу <Win> на клавиатуре

Четвертый аргумент функции задает код виртуальной клавиши (см. табл. 3.19). При успешном выполнении функции будет возвращено ненулевое значение. Если комбинация клавиш уже была зарегистрирована в системе другой программой, функция возвратит ошибку. В некоторых системах (например, в Windows 2000) клавиша <F12> занята системой и ее не стоит использовать в качестве "горячей" клавиши. После регистрации всех необходимых программе "горячих" клавиш нужно добавить в функцию главного окна программы обработчик сообщения WM_HOTKEY. В нем будет сосредоточена вся работа по обработке зарегистрированных клавиш. После завершения работы с программой следует для каждой "горячей" клавиши вызвать функцию UnregisterHotKey, которая удалит регистрацию клавиши из системы. Чтобы закрепить полученные сведения, изучите листинг 3.28.

Листинг 3.28. Использование "горячих" клавиш в программе

```
// в начале файла описываем свои клавиши
#define MY_HOT_KEY_1   WM_USER + 5001
#define MY_HOT_KEY_2   WM_USER + 5002
#define MY_HOT_KEY_3   WM_USER + 5003
```

```
// в главную функцию окна добавляем поддержку клавиш
LRESULT CALLBACK  MyMainProc ( HWND hWnd, UINT message,
                               UINT wParam, LONG lParam)
{
      switch ( message)
        {
        case WM_CREATE: // во время создания окна регистрируем клавиши
          // для первой клавиши назначаем комбинацию Ctrl+F2
          RegisterHotKey ( hWnd, MY_HOT_KEY_1, MOD_CONTROL, VK_F2);
          // для второй клавиши назначаем Home
          RegisterHotKey ( hWnd, MY_HOT_KEY_2, NULL, VK_HOME);
          // для третьей клавиши назначаем комбинацию Ctrl+Shift+F5
          RegisterHotKey ( hWnd, MY_HOT_KEY_3, MOD_CONTROL | MOD_SHIFT,
                                                             VK_F5);
          break;
        case WM_HOTKEY: // обработка "горячих" клавиш
          {
           switch ( wParam) // wParam содержит идентификатор
            {
             case MY_HOT_KEY_1:
             // выполняем какую-либо свою команду
               break;
             case MY_HOT_KEY_2:
             // выполняем какую-либо свою команду
               break;
             case MY_HOT_KEY_3:
             // выполняем какую-либо свою команду
               break;
            }
          }
          break;
        case WM_DESTROY: // завершаем программу
        // удаляем регистрацию клавиш
        UnregisterHotKey ( 0, MY_HOT_KEY_1);
        UnregisterHotKey ( 0, MY_HOT_KEY_2);
        UnregisterHotKey ( 0, MY_HOT_KEY_3);
          break;
        default:
          return ( DefWindowProc ( hWnd, message, wParam, lParam));
        }
 return ( 0);
 }
```

3.4.3. Поддержка языков

Как правило, на обычном компьютере устанавливается поддержка не только русского, но и английского (или любого другого) или нескольких языков. Для переключения между ними применяется предопределенная комбинация клавиш. Мы можем из программы управлять выбором того или иного языка. За каждым поддерживаемым языком закреплен идентификационный номер. Для загрузки нового языка применяется функция LoadKeyboardLayout. Первый аргумент функции указывает на идентификатор языка (строка с шестнадцатеричным значением), который следует загрузить. Например, для английского языка идентификационный номер равен 0x0409, где младшее слово указывает на идентификатор языка, а старшее — на дескриптор устройства (драйвера поддержки языка). В функцию записывается строковое значение "00000409". Для русского языка идентификационный номер будет равен "00000419" (0x0419). Второй аргумент функции определяет дополнительный флаг, который может принимать одно из перечисленных в табл. 3.21 значений.

***Таблица 3.21.** Значение флага для раскладки клавиатуры*

Значение флага	Описание
KLF_REORDER	Передвигает раскладку данного языка на первое место в списке
KLF_REPLACELANG	Если указанный идентификатор совпадает с уже имеющимся, то он все равно заменяет последний
KLF_ACTIVATE	Делает активной загруженную раскладку языка

При успешном выполнении функция возвратит идентификатор (тип HKL) загруженного языка. Для того чтобы выгрузить какой-либо язык, следует использовать функцию UnloadKeyboardLayout. Она имеет только один аргумент, указывающий на идентификатор языка. При успешном выполнении функция возвратит ненулевое значение. Дополнительно к описанным функциям можно применять ActivateKeyboardLayout для выбора текущего языка. Функция содержит два аргумента. Первый указывает на идентификатор языка, но может также принимать одно из следующих значений: HKL_NEXT — выбрать следующий язык из списка, HKL_PREV — выбрать предыдущий язык из списка. Второй аргумент определяет флаг операции. Это может быть одно из следующих значений: KLF_REORDER — переопределить список с учетом выбранного, KLF_RESET — сбросить блокировку использования заглавных букв (только для Windows 2000), KLF_SHIFTLOCK — включить блокировку использования заглавных букв (только для Windows 2000). В листинге 3.29 приведен пример работы с рассмотренными функциями.

Листинг 3.29. Работа с раскладкой клавиатуры

```
// загружаем русскую раскладку клавиатуры
HKL hRus = LoadKeyboardLayout ( "00000419", KLF_REORDER);
// делаем русский язык текущим
if ( hRus != NULL)
ActivateKeyboardLayout ( hRus, KLF_REORDER);
// после работы с русской раскладкой, выгружаем ее
UnloadKeyboardLayout ( hRus);
```

Существует также возможность получения информации о текущей раскладке клавиатуры. Для этого используются функции GetKeyboardLayout и GetKeyboardLayoutName. Первая функция возвращает числовой идентификатор (тип HKL) текущего языка, а вторая — строку с именем раскладки (например, "00000419").

На этом мне бы хотелось завершить описание различных способов программирования клавиатуры и перейти к рассмотрению не менее важного элемента любого компьютера — видеоадаптера.

ГЛАВА 4

Видеоадаптер

Видеоадаптер является, пожалуй, одним из самых интересных устройств, входящих в состав современного компьютера. В тандеме с монитором он позволяет проникнуть в совершенно другой, виртуальный мир, расцвеченный невообразимыми красками и эффектами. Он служит своеобразным мостом между бездушными железками и ожившими образами. Вряд ли компьютер получил бы такое широкое признание людей, далеких от техники, если бы не его способность приоткрывать дверь в другой, так не похожий на наш, мир.

Это небольшое отступления я сделал не для того, чтобы повторить простые истины, но показать, насколько большое значение имеет видеоадаптер и монитор в общении простого пользователя с компьютером. Из сказанного выше разработчик программного обеспечения должен сделать один главный вывод: необходимо знать и уметь программировать возможности видеоадаптера и монитора. Не стоит всецело полагаться на готовый визуальный интерфейс Windows или библиотеку DirectX, многие задачи (например, создание динамичных игрушек) потребуют более широких знаний и умений работать с оборудованием.

В этой главе мы рассмотрим различные возможности доступа к видеоадаптеру (видеоконтроллеру), которые можно представить в виде списка:

1. Использование функций BIOS через прерывание `int 10h`.
2. Использование аппаратных портов.
3. Использование возможностей Win32 API.

4.1. Общие сведения

Стандарты видеоадаптеров имеют достаточно большую историю. Первые монохромные адаптеры появились в начале 80-х годов прошлого века. Они

поддерживали только два цвета (обычно зеленый на черном фоне) и работали исключительно в текстовом режиме. В этом режиме на экран монитора с разрешением 80×25 (столбцов×строк) выводился стандартного размера прямоугольник (14 на 9 пикселов) с определенным символом. Далее появился новый стандарт — CGA (Color Graphics Adapter), позволяющий работать как в текстовом, так и в графическом режиме. В последнем режиме он поддерживал два разрешения: 320×200 (4 цвета) и 640×200 (2 цвета) пикселов. Были и другие стандарты (HGC, EGA), давно устаревшие и не используемые в современных видеоадаптерах.

Сегодня применяются современные VGA-совместимые (Video Graphics Array) видеоадаптеры, которые поддерживают большое количество текстовых и графических режимов: 4 текстовых, 256-цветный, HiColor (65 536 цветов) и TrueColor (минимум 16,7 млн цветов). Для унификации управления режимами видеоадаптеров был разработан общий стандарт VBE (VESA BIOS EXTENSION). Он позволил определить одинаковые способы доступа к основным функциям многочисленных графических адаптеров различных фирм. Базируясь на наборе функций BIOS, использующих прерывание `int 10h`, он может применяться для программирования любых VBE-совместимых видеоадаптеров. Прежде чем рассмотреть возможности этого стандарта, хочу сказать два слова о видеопамяти. Каждый видеоадаптер содержит определенный объем памяти, который применяется для промежуточного хранения данных, поступающих от компьютера. От размера видеопамяти напрямую зависят возможности самого адаптера: поддерживаемое разрешение и скорость работы. Существует простая формула, позволяющая подсчитать необходимый размер памяти для работы в том или ином разрешении. Для этого нужно перемножить значения разрешения по вертикали и горизонтали и умножить результат на количество байт в одном пикселе (16,7 млн — 3 байта, 65 536 — 2 байта, 256 — 1 байт). Например, для поддержки разрешения 1024×768×16 размер памяти должен быть как минимум 2 Мбайт. А теперь непосредственно приступим к изучению набора функций BIOS для программирования видеоадаптеров.

4.2. Использование функций BIOS

Существует два набора функций BIOS: для работы с VGA и Super VGA режимами. Большинство режимов VGA устарели, но для полноты информации приведу список основных режимов в табл. 4.1. Кроме них, существует еще много других режимов, поддерживаемых производителями видеоконтроллеров.

Для поддержки работы с видеоадаптером дополнительно в системной памяти (в первом мегабайте) выделяется окно определенного размера. Для текстовых режимов размер окна равен 32 Кбайт, а диапазон адресов — `B8000h`—`B7FFFh` (за исключением режимов `04h` и `06h`). Для графических режимов раз-

мер окна составляет 64 Кбайт, а диапазон — A0000h—AFFFFh. Именно через это окно происходит запись данных в память видеоадаптера. При работе в текстовом режиме каждому символу в памяти выделяется 2 байта: первый хранит ASCII-код символа, а второй — атрибуты символа (см. табл. 4.2). Для чего нужен байт атрибутов, вы узнаете дальше, по ходу изучения функций BIOS.

Таблица 4.1. *Основные режимы VGA*

Код	Разрешение	Количество цветов	Число страниц
00h	40 × 25 (текстовый)	16	8
01h	40 × 25 (текстовый)	16	8
02h	80 × 25 (текстовый)	16	8
03h	80 × 25 (текстовый)	16	8
04h	320 × 200 (графический)	4	4
05h	320 × 200 (графический)	4	4
06h	640 × 200 (графический)	2	2
07h	80 × 25 (текстовый)	2	—
0Dh	320 × 200 (графический)	16	8
0Eh	640 × 200 (графический)	16	4
0Fh	640 × 350 (графический)	2	2
10h	640 × 350 (графический)	16	—
11h	640 × 480 (графический)	2	—
12h	640 × 480 (графический)	16	—
13h	320 × 200	256	—

4.2.1. Функция *00h*

Эта функция позволяет установить текстовый или графический режим. Как правило, эту функцию следует вызывать в самом начале программы, независимо от текущего режима. Если в программе использовался графический режим, желательно перед ее завершением переключиться в текстовый.

Использование:

1. В регистр AH следует поместить код функции 00h.
2. В регистр AL нужно записать код режима (см. табл. 4.1).
3. Вызвать прерывание int 10h.

Выход:

После выполнения функции некоторые BIOS (например, AMI BIOS) запишут в регистр `AL` определенное значение:

- `20h` — код установленного режима больше `05h`;
- `30h` — код установленного режима лежит в диапазоне `0h—5h` или равен `07h`;
- `3Fh` — код режима равен `06h`.

С помощью данной функции можно очистить экран дисплея. Пример установки текстового режима показан в листинге 4.1.

Листинг 4.1. Использование функции 00h

```
mov AH, 00h    ; код функции 00h
mov AL, 1      ; текстовый режим 40 x 25 x 16
int 10h        ; вызываем прерывание
cmp Al, 30h    ; проверяем значение для AMI BIOS
jne Error_Mode; вызываем обработчик, если режим не установлен
```

4.2.2. Функция *01h*

Функция позволяет установить размер курсора для текстовых режимов работы видеоадаптера. Курсор служит удобным указателем точки ввода на экране дисплея. Его размер может варьироваться от линии до полного размера прямоугольника, отведенного под один символ. Курсор состоит из горизонтальных линий.

Использование:

1. В регистр `AH` следует поместить код функции `01h`.
2. В регистр `CH` нужно записать начальную линию, начиная сверху. Для хранения значения линии используются только биты с 0 по 4. Некоторые BIOS могут поддерживать дополнительно установку общего вида курсора. При этом бит 7 должен быть установлен в 0, а биты 5 и 6 — в одно из следующих значений: `00h` — обычный вид, `01h` — невидимый. Еще раз повторю, что установка битов 5 и 6 работает не на всех системах, поэтому желательно биты 5, 6 и 7 устанавливать в 0.
3. В регистр `CL` нужно записать конечную линию.
4. Вызвать прерывание `int 10h`.

Выход:

Возвращаемого значения нет. Рассмотрим простой пример установки размера курсора, состоящего из 10 линий (листинг 4.2).

Листинг 4.2. Использование функции 01h

```
mov AH, 01h   ; код функции 01h
mov CH, 0     ; номер начальной линии, начиная сверху экрана
mov CL, 0Ah   ; номер конечной линии равен 10
int 10h       ; вызываем прерывание
```

4.2.3. Функция *02h*

Данная функция позволяет установить текущую позицию курсора на экране дисплея. Отсчет координат производится с левого верхнего угла (0, 0). Кроме того, следует учитывать номер текущей страницы видеопамяти, который может меняться для различных режимов от 0 до 8. Каждая страница содержит свой курсор, поэтому при переключении страниц установку позиции курсора необходимо выполнять заново. Если используется монохромный дисплей (бывает и такое), номер страницы памяти всегда должен быть равен 0. Номер страницы следует устанавливать в 0 и при работе в графических режимах.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. В регистр BH нужно записать номер страницы.
3. В регистр DH нужно записать номер строки (от 00h).
4. В регистр DL нужно записать номер столбца (от 00h).
5. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Рассмотрим простой пример установки позиции курсора на 2-ю строку 27-го столбца в листинге 4.3.

Листинг 4.3. Использование функции 02h

```
mov AH, 02h   ; код функции 02h
mov BH, 0     ; номер страницы
mov DH, 2     ; номер строки
mov DL, 27    ; номер столбца
int 10h       ; вызываем прерывание
```

Данная функция также позволяет скрыть курсор с экрана дисплея. Для этого достаточно установить значение номера строки на единицу больше, чем максимальное разрешение по вертикали (для текстовых режимов 25). Не забудьте, что отсчет ведется с 0. Пример кода, позволяющий убрать курсор с экрана дисплея, будет выглядеть так, как показано в листинге 4.4.

Листинг 4.4. Использование функции 02h для скрытия курсора

```
mov AH, 02h   ; код функции 02h
mov BH, 0     ; номер страницы
mov DH, 25    ; номер строки
mov DL, 0     ; номер столбца
int 10h       ; вызываем прерывание
```

4.2.4. Функция *03h*

Данная функция позволяет получить размер и позицию курсора для указанной страницы видеопамяти.

Использование:

1. В регистр AH следует поместить код функции 03h.
2. В регистр BH нужно записать номер страницы видеопамяти (от 0 до 7).
3. Вызвать прерывание int 10h.

Выход:

После выполнения функции интересующая нас информация будет распределена следующим образом:

- регистр AX — значение 0000h (только для Phoenix BIOS);
- регистр CH — номер начальной линии курсора;
- регистр CL — номер конечной линии курсора;
- регистр DH — номер строки позиции курсора;
- регистр DL — номер столбца позиции курсора.

Пример кода, позволяющий определить позицию для текущего курсора, представлен в листинге 4.5.

Листинг 4.5. Использование функции 03h

```
mov AH, 03h   ; код функции 03h
mov BH, 0     ; номер страницы
int 10h       ; вызываем прерывание
; получаем позицию курсора
mov CUR_Y, DH; номер строки
mov CUR_X, DL; номер столбца
```

4.2.5. Функция *05h*

Функция позволяет выбрать текущую страницу видеопамяти для текстового режима. Диапазон возможных значений лежит от 00h до 07h.

Использование:

1. В регистр AH следует поместить код функции 05h.
2. В регистр AL нужно записать номер страницы видеопамяти.
3. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Рассмотрим простой пример, в котором выберем текущую страницу под номером 2 (листинг 4.6).

Листинг 4.6. Использование функции 05h

```
mov AH, 05h   ; код функции 05h
mov BH, 01h   ; вторая страница видеопамяти
int 10h       ; вызываем прерывание
```

4.2.6. Функция *06h*

Эта функция позволяет прокрутить активную страницу вверх. Перед вызовом данной функции рекомендуется установить номер активной страницы с помощью предыдущей функции 05h.

Использование:

1. В регистр AH следует поместить код функции 06h.
2. В регистр AL нужно записать число строк, на которое будет выполнена прокрутка. Если установить значение 00h, указанная часть страницы будет полностью очищена.
3. В регистр BH нужно записать значение байта атрибутов.
4. В регистр CH следует поместить номер строки верхнего левого угла окна.
5. В регистр CL следует поместить номер столбца верхнего левого угла окна.
6. В регистр DH следует поместить номер строки нижнего правого угла окна.
7. В регистр DL следует поместить номер столбца нижнего правого угла окна.
8. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Данную функцию можно с успехом использовать для очистки всего или части экрана. Формат байта атрибутов показан в табл. 4.2. Например, чтобы прокрутить экран вверх на 3 строки, следует написать код, показанный в листинге 4.7.

Листинг 4.7. Использование функции 06h

```
mov AH, 05h  ; код функции 05h
mov BH, 00h  ; выбираем первую страницу
int 10h      ; вызываем прерывание
mov AH, 06h  ; код функции 06h
mov AL, 03h  ; число строк
mov BH, 11   ; атрибут очистки
mov CH, 0    ; номер строки верхнего угла
mov CL, 0    ; номер столбца верхнего угла
mov DH, 24   ; номер строки нижнего угла
mov CL, 79   ; номер столбца нижнего угла
int 10h      ; вызываем прерывание
```

Таблица 4.2. *Формат байта атрибутов*

Бит	Описание
0	Использование синего в основном цвете (1 — использовать)
1	Использование зеленого в основном цвете (1 — использовать)
2	Использование красного в основном цвете (1 — использовать)
3	Повышенная насыщенность (1 — использовать)
4	Использование синего в фоновом цвете (1 — использовать)
5	Использование зеленого в фоновом цвете (1 — использовать)
6	Использование красного в фоновом цвете (1 — использовать)
7	Использовать мигание

Код, позволяющий очистить экран, представлен в листинге 4.8.

Листинг 4.8. Использование функции 06h для очистки экрана

```
mov AH, 06h  ; код функции 06h
mov AL, 00h  ; число строк обнуляем
mov BH, 7    ; атрибут очистки
mov CH, 2    ; номер строки верхнего угла
mov CL, 15   ; номер столбца верхнего угла
mov DH, 10   ; номер строки нижнего угла
mov CL, 15   ; номер столбца нижнего угла
int 10h      ; вызываем прерывание для очистки окна
```

4.2.7. Функция *07h*

Функция позволяет прокрутить активную страницу вниз. Перед вызовом данной функции рекомендуется установить номер активной страницы с помощью предыдущей функции 05h.

Использование:

1. В регистр AH следует поместить код функции 07h.
2. В регистр AL нужно записать число строк, на которое будет выполнена прокрутка. Если установить значение 00h, указанная часть страницы будет полностью очищена.
3. В регистр BH нужно записать значение байта атрибутов (см. табл. 4.2).
4. В регистр CH следует поместить номер строки верхнего левого угла окна.
5. В регистр CL следует поместить номер столбца верхнего левого угла окна.
6. В регистр DH следует поместить номер строки нижнего правого угла окна.
7. В регистр DL следует поместить номер столбца нижнего правого угла окна.
8. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Данную функцию можно с успехом использовать для очистки всего или части экрана. Например, чтобы прокрутить экран вниз на 4 строки, следует написать код, как в листинге 4.9.

Листинг 4.9. Использование функции 07h

```
mov AH, 05h   ; код функции 05h
mov BH, 00h   ; выбираем первую страницу
int 10h       ; вызываем прерывание
mov AH, 07h   ; код функции 07h
mov AL, 04h   ; число строк
mov BH, 7     ; атрибут очистки
mov CH, 0     ; номер строки верхнего угла
mov CL, 0     ; номер столбца верхнего угла
mov DH, 24    ; номер строки нижнего угла
mov CL, 79    ; номер столбца нижнего угла
int 10h       ; вызываем прерывание
```

Как видите, функция 07h аналогична 06h и также может использоваться для очистки экрана или сдвига экрана в сторону.

4.2.8. Функция *08h*

Данная функция предназначена для чтения символа и его атрибутов для текущей позиции курсора.

Использование:

1. В регистр AH следует поместить код функции 08h.
2. В регистр BH нужно записать номер страницы.
3. Вызвать прерывание int 10h.

Выход:

После выполнения функции в регистр AH будет записан атрибут символа, а в регистр AL — код символа. Атрибут символа (см. табл. 4.2) характеризует следующие свойства: цвет фона (биты 4—6), цвет символа (биты 0—2), насыщенность (бит 3) и мигание (бит 7). Возможные значения цвета символа представлены в табл. 4.3.

Таблица 4.3. *Набор значений цвета для символов*

Значения битов цвета (0—2)			Насыщенность (бит 3)	
Синий (бит 0)	Зеленый (бит 1)	Красный (бит 2)	Включена (1)	Выключена (0)
0	0	0	Серый	Черный
0	0	1	Светло-синий	Синий
0	1	0	Светло-зеленый	Зеленый
0	1	1	Светлый циан	Циан
1	0	0	Светло-красный	Красный
1	0	1	Светло-фиолетовый	Фиолетовый
1	1	0	Желтый	Коричневый
1	1	1	Белый	Грязно-белый

Возможные значения цвета для фона имеют такие же значения, но для битов 4—6. Кроме того, бит 7, а не 3 определяет насыщенность для цветовых комбинаций фона. Небольшой пример кода для получения атрибутов символа показан в листинге 4.10.

Листинг 4.10. Использование функции 08h

```
; установим прежде позицию курсора
mov AH, 02h    ; функция 02h
mov BH, 00h    ; номер страницы
```

```
mov DH, 5        ; номер строки
mov DL, 33       ; номер столбца
int 10h          ; вызываем прерывание
mov AH, 08h      ; код функции 08h
mov BH, 00h      ; выбираем первую страницу
int 10h          ; читаем атрибуты символа для текущей позиции курсора
mov CH_ATR, AH; сохраняем атрибуты символа в переменной CH_ATR
and AH, 01111b; выделяем значение цвета символа
mov CH_CLR, AH; сохраняем значение цвета, если нужно
cmp AH, 14       ; можем проверить наличие желтого цвета у символа
jne Error_CLR ; цвет символа не желтый
```

4.2.9. Функция *09h*

Данная функция предназначена для записи символа с указанными атрибутами в текущую позицию курсора.

Использование:

1. В регистр AH следует поместить код функции 09h.
2. В регистр AL нужно записать символ.
3. В регистр BH следует поместить номер страницы.
4. В регистр BL нужно записать байт атрибутов для символа. В графическом режиме сюда необходимо записать значение цвета.
5. В регистр CX нужно поместить число повторений для записи символа.
6. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. С помощью этой функции можно также очистить экран дисплея в текстовом режиме. Пример использования функции 09h представлен в листинге 4.11.

Листинг 4.11. Использование функции 09h

```
; установим прежде позицию курсора
mov AH, 02h   ; функция 02h
mov BH, 00h   ; номер страницы
mov DH, 1     ; номер строки
mov DL, 10    ; номер столбца
int 10h       ; вызываем прерывание
mov AH, 9     ; код функции 09h
mov AL, 'w'   ; указываем символ, который нужно записать
mov BH, 00h   ; номер страницы
```

```
mov BL, 14    ; желтый цвет символа
mov CX, 5     ; повторить 5 раз
int 10h       ; вызываем прерывание
```

Для очистки экрана в текстовом режиме можно использовать код, показанный в листинге 4.12.

Листинг 4.12. Использование функции 09h для очистки экрана

```
mov AH, 02h   ; функция 02h
mov BH, 00h   ; номер страницы
xor DX, DX    ; верхний левый угол окна (0, 0)
int 10h       ; вызываем прерывание
mov AH, 9     ; код функции 09h
mov AL, ' '   ; указываем символ пробела
mov BH, 00h   ; номер страницы
mov BL, 0     ; байт атрибутов
mov CX, 2000 ; число повторов (80 * 25)
int 10h       ; вызываем прерывание
```

4.2.10. Функция *0Ah*

Данная функция предназначена для записи символа в текущую позицию курсора. Отличается от функции 09h отсутствием установки байта атрибута.

Использование:

1. В регистр AH следует поместить код функции 0Ah.
2. В регистр AL нужно записать символ.
3. В регистр BH следует поместить номер страницы.
4. В регистр CX нужно поместить число повторений для записи символа.
5. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Поскольку функция практически ничем не отличается от 09h, пример приводить не буду.

4.2.11. Функция *0Bh*

Данная функция имеет несколько подфункций, выполняющих различные операции. Рассмотрим их подробнее.

4.2.11.1. Подфункция *00h*

Подфункция предназначена для установки цвета обрамляющей экран рамки в текстовом режиме и цвета фона в графическом режиме.

Использование:

1. В регистр AH следует поместить код функции 0Bh.
2. В регистр BH следует поместить код подфункции 00h.
3. В регистр BL нужно поместить значение цвета (от 0 до 15).
4. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Пример кода, позволяющий в режиме 04h (см. табл. 4.1) установить желтый фон, показан в листинге 4.13.

Листинг 4.13. Использование функции 0B00h

```
mov AH, 0     ; установим новый режим
mov AL, 04h   ; графический режим 04h
int 10h       ; вызываем прерывание
mov AH, 0Bh   ; код функции 0Bh
mov BH, 00h   ; код подфункции 00h
mov BL, 14    ; установим желтый цвет фона
int 10h       ; вызываем прерывание
```

4.2.11.2. Подфункция *01h*

Эта подфункция позволяет установить текущую палитру для 04h или 05h режимов. Палитра устанавливает три основных цвета для отображаемых символов. Переключение с одной палитры на другую позволяет быстро изменить используемые цвета.

Использование:

1. В регистр AH следует поместить код функции 0Bh.
2. В регистр BH следует поместить код подфункции 01h.
3. В регистр BL нужно поместить значение код палитры. Код палитры может иметь одно из двух значений: 00h (зеленый, красный, коричневый/желтый) или 01h (циан, фиолетовый, белый).
4. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Чтобы установить палитру для графического режима, можно использовать код, представленный в листинге 4.14.

Листинг 4.14. Использование функции 0B01h

```
mov AH, 0     ; установим новый режим
mov AL, 04h   ; режим 04h
```

```
int 10h       ; вызываем прерывание
mov AH, 0Bh   ; код функции 0Bh
mov BH, 00h   ; код подфункции 01h
mov BL, 00h   ; установим палитру с кодом 00h
int 10h       ; вызываем прерывание
```

4.2.12. Функция *0Ch*

Функция позволяет вывести на экран дисплея один пиксел. Данную функцию следует использовать только в графических режимах.

Использование:

1. В регистр AH следует поместить код функции 0Ch.
2. В регистр AL следует поместить значение цвета для пиксела. Если бит 7 установить в 1, будет выполнено преобразование XOR (исключающее ИЛИ) над устанавливаемым цветом и текущим.
3. В регистр CX нужно поместить номер столбца.
4. В регистр DX требуется записать номер строки.
5. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Пример кода, позволяющий вывести на экран желтую точку, показан в листинге 4.15.

Листинг 4.15. Использование функции 0Ch

```
mov AH, 0     ; установим новый режим
mov AL, 04h   ; графический режим 04h, 320 x 200
int 10h       ; вызываем прерывание
mov AH, 0Ch   ; код функции 0Ch
mov AL, 14    ; установим желтый цвет
mov CX, 190   ; столбец 190
mov DX, 150   ; строка 150
int 10h       ; вызываем прерывание
```

4.2.13. Функция *0Dh*

Данная функция позволяет прочитать с экрана дисплея один пиксел. Данную функцию следует использовать только в графических режимах.

Использование:

1. В регистр AH следует поместить код функции 0Dh.
2. В регистр CX нужно поместить номер столбца.

3. В регистр DX требуется записать номер строки.
4. Вызвать прерывание int 10h.

Выход:

После выполнения функции регистр AL будет содержать требуемый пиксел. Пример работы с данной функцией представлен в листинге 4.16.

Листинг 4.16. Использование функции 0Dh

```
mov AH, 0    ; установим новый режим
mov AL, 04h  ; графический режим 04h, 320 x 200
int 10h      ; вызываем прерывание
mov AH, 0Dh  ; код функции 0Dh
mov CX, 174  ; столбец 174
mov DX, 100  ; строка 100
int 10h      ; вызываем прерывание
mov PX_1, AL ; сохраняем значение пиксела в переменной PX_1
```

4.2.14. Функция *0Eh*

Функция позволяет выводить символы на экран дисплея в режиме телетайпа. Эту функцию удобно использовать для вывода строки символов вместо функции 09h.

Использование:

1. В регистр AH следует поместить код функции 0Eh.
2. В регистр AL нужно поместить символ.
3. В регистр BH нужно поместить номер страницы.
4. В регистр BL требуется записать цвет фона (для графического режима).
5. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. После выполнения функция автоматически передвигает курсор и прокручивает (если необходимо) экран. Использование управляющих символов (07h, 08h, 0Ah и 0Dh) приведет к результату, определяемому их стандартным назначением. Пример работы с данной функцией представлен в листинге 4.17.

Листинг 4.17. Использование функции 0Eh

```
my_msg db "Тестовая строка $"
; код программы
mov AH, 0Eh   ; код функции 0Eh
```

```
lea BX, my_msg; получаем адрес строки
@repeat:
mov AL, [BX]  ; записываем первый символ
cmp AL, '$'   ; проверяем конец строки
je  EXIT_CODE ; выходим из цикла
int 10h       ; выводим символ на экран
inc BX        ; увеличиваем регистр на 1
jmp short @repeat
```

4.2.15. Функция *0Fh*

Эта функция позволяет получить текущий видеорежим.

Использование:

1. В регистр AH следует поместить код функции 0Fh.
2. Вызвать прерывание int 10h.

Выход:

После успешного выполнения функции в регистр AL будет помещен код текущего видеорежима. В регистр AH будет записано число символов в одной строке, а в регистр BH — номер активной страницы. Чтобы определить текущий режим дисплея, можно использовать код, показанный в листинге 4.18.

Листинг 4.18. Использование функции 0Fh

```
mov AH, 0Fh  ; код функции 0Fh
int 10h      ; вызываем прерывание
cmp AL, 00h  ; если текстовый 40 x 25
je  TXT_MODE ; передаем управление
cmp AL, 04h  ; если графический 320 x 200
je  GRAF_MODE; передаем управление
```

4.2.16. Функция *12h*

Функция содержит дополнительные подфункции для выполнения различных настроек. Поддержка BIOS данного набора подфункций необязательна.

4.2.16.1. Подфункция *30h*

Данная подфункция позволяет установить вертикальное разрешение для текстовых режимов. Установленное разрешение вступит в силу только после следующей установки режима дисплея.

Использование:

1. В регистр AH следует поместить код функции 12h.
2. В регистр BL нужно записать код подфункции 30h.
3. В регистр AL следует поместить значение разрешения. Возможны следующие варианты: 00h — 200 строк, 01h — 350 строк, 02h — 400 строк.
4. Вызвать прерывание int 10h.

Выход:

Если данная подфункция поддерживается, в регистр AL будет записано значение 12h. Пример кода работы с данной подфункцией приведен в листинге 4.19.

Листинг 4.19. Использование функции 1230h

```
mov AH, 12h   ; код функции 12h
mov BL, 30h   ; код подфункции 30h
mov AL, 02h   ; установим 400 строк по вертикали
int 10h       ; вызываем прерывание
cmp AL, 12h   ; проверим результат
jne ERROR_HND; функция не поддерживается, переходим к обработчику ошибок
```

4.2.16.2. Подфункция *31h*

Подфункция позволяет загрузить палитру, используемую по умолчанию.

Использование:

1. В регистр AH следует поместить код функции 12h.
2. В регистр BL нужно записать код подфункции 31h.
3. В регистр AL следует поместить код операции. Возможны следующие варианты: 00h — разрешить загрузку палитры по умолчанию, 01h — запретить загрузку палитры по умолчанию.
4. Вызвать прерывание int 10h.

Выход:

Если данная подфункция поддерживается, в регистр AL будет записано значение 12h. Пример кода работы с данной подфункцией приводится в листинге 4.20.

Листинг 4.20. Использование функции 1231h

```
mov AH, 12h   ; код функции 12h
mov BL, 31h   ; код подфункции 31h
```

```
mov AL, 00h  ; разрешить загрузку палитры по умолчанию
int 10h      ; вызываем прерывание
cmp AL, 12h  ; проверим результат
jne ERROR_HND; функция не поддерживается, переходим к обработчику ошибок
```

4.2.16.3. Подфункция *32h*

Эта подфункция управляет доступом процессора к видеопамяти и регистрам ввода-вывода.

Использование:

1. В регистр AH следует поместить код функции 12h.
2. В регистр BL нужно записать код подфункции 32h.
3. В регистр AL следует поместить код операции. Возможны следующие варианты: 00h — разрешить доступ, 01h — запретить доступ.
4. Вызвать прерывание int 10h.

Выход:

Если данная подфункция поддерживается, в регистр AL будет записано значение 12h. Пример кода работы с данной подфункцией приводится в листинге 4.21.

Листинг 4.21. Использование функции 1232h

```
mov AH, 12h  ; код функции 12h
mov BL, 32h  ; код подфункции 32h
mov AL, 01h  ; запретить доступ к видеоконтроллеру
int 10h      ; вызываем прерывание
cmp AL, 12h  ; проверим результат
jne ERROR_HND; функция не поддерживается, переходим к обработчику ошибок
```

4.2.16.4. Подфункция *36h*

Подфункция позволяет управлять регенерацией (обновлением) экрана дисплея.

Использование:

1. В регистр AH следует поместить код функции 12h.
2. В регистр BL нужно записать код подфункции 36h.
3. В регистр AL следует поместить код операции. Возможны следующие варианты: 00h — разрешить обновление, 01h — запретить обновление.
4. Вызвать прерывание int 10h.

Выход:

Если данная подфункция поддерживается, в регистр AL будет записано значение 12h. Пример кода работы с данной подфункцией приводится в листинге 4.22.

Листинг 4.22. Использование функции 1236h

```
mov AH, 12h  ; код функции 12h
mov BL, 36h  ; код подфункции 36h
mov AL, 01h  ; запретить регенерацию экрана дисплея
int 10h      ; вызываем прерывание
cmp AL, 12h  ; проверим результат
jne ERROR_HND; функция не поддерживается, переходим к обработчику ошибок
```

4.2.17. Функция *1000h*

Данная функция предназначена для установки одного регистра палитры.

Использование:

1. В регистр AX следует поместить код функции 1000h.
2. В регистр BL нужно записать номер регистра палитры (00h—0Fh).
3. В регистр BH следует поместить желаемый цвет.
4. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.18. Функция *1001h*

Функция позволяет установить цвет обрамляющей экран рамки.

Использование:

1. В регистр AX следует поместить код функции 1001h.
2. В регистр BH следует поместить желаемый цвет (00h—3Fh). Значение 00h соответствует черному цвету.
3. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.19. Функция *1002h*

Функция позволяет одновременно записать значения для всех регистров палитры.

Использование:

1. В регистр AX следует поместить код функции 1002h.
2. В регистр ES:DX следует поместить указатель на список регистров палитры. Список регистров должен иметь размер 17 байт, где 16-й байт определяет цвет обрамляющей рамки экрана, а байты с 0 по 15 задают цвета палитры.
3. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.20. Функция *1003h*

Эта функция позволяет управлять битом 7 в байте атрибутов. Данный бит отвечает за мигание символа и насыщенность.

Использование:

1. В регистр AX следует поместить код функции 1003h.
2. В регистр BL следует поместить код операции. Возможны следующие варианты: 00h — включить насыщенность, 01h — включить мигание.
3. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.21. Функция *1007h*

Функция позволяет получить определенный регистр палитры.

Использование:

1. В регистр AX следует поместить код функции 1007h.
2. В регистр BL следует поместить номер регистра палитры (00h—0Fh).
3. Вызвать прерывание int 10h.

Выход:

При успешном выполнении в регистр BH будет записан цвет для указанного регистра палитры.

4.2.22. Функция *1008h*

Функция позволяет определить текущий цвет обрамляющей рамки экрана.

Использование:

1. В регистр AX следует поместить код функции 1008h.
2. Вызвать прерывание int 10h.

Выход:

При успешном выполнении в регистр BH будет записано значение цвета (00h—3Fh) обрамляющей рамки экрана.

4.2.23. Функция *1009h*

Данная функция позволяет одновременно прочитать все регистры палитры и регистр обрамляющей рамки экрана.

Использование:

1. В регистр AX следует поместить код функции 1009h.
2. В регистр ES:DX следует поместить указатель на массив из 17 байтов.
3. Вызвать прерывание int 10h.

Выход:

После выполнения функции выделенный массив будет заполнен данными о цветах палитры (от 0 до 15 байта) и значением цвета обрамляющей рамки (байт 16). Функция проста и не требует примера.

4.2.24. Функция *1010h*

Функция позволяет установить значения насыщенности цветов для одного регистра ЦАП (цифроаналоговый преобразователь).

Использование:

1. В регистр AX следует поместить код функции 1010h.
2. В регистр BX надо записать номер регистра ЦАП (от 00h до FFh).
3. В регистр CH требуется занести новое значение насыщенности для зеленого цвета (от 0 до 63).
4. В регистр CL требуется занести новое значение насыщенности для синего цвета (от 0 до 63).
5. В регистр DH требуется занести новое значение насыщенности для красного цвета (от 0 до 63).
6. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.25. Функция *1012h*

Эта функция позволяет установить значения насыщенности цветов для целой группы регистров ЦАП.

Использование:

1. В регистр AX следует поместить код функции 1012h.
2. В регистр BX надо записать номер первого регистра ЦАП (от 00h до FFh) в группе.
3. В регистр CH требуется занести новое значение насыщенности для зеленого цвета (от 0 до 63).
4. В регистр CX требуется записать количество регистров в группе.
5. В регистр ES:DX следует поместить указатель на список описателей для каждого регистра группы. Размер каждого описателя равен трем байтам (красный, синий, зеленый). Значение каждого байта цвета лежит в промежутке от 0 до 63. Общий размер всей группы должен быть равен 3*CX байтов.
6. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.26. Функция *1013h*

Функция позволяет выбрать страницу видеопамяти для регистров цвета ЦАП. Данная функция не поддерживает графический режим 13h.

Использование:

1. В регистр AX следует поместить код функции 1013h.
2. В регистр BL надо записать тип операции: 00h — количество страниц, 01h — номер страницы.
3. В регистр BH требуется записать одно из следующих значений: для первого типа надо указать количество страниц (00h — 4, 01h — 16), для второго следует определить номер страницы (00h—03h или 00h—0Fh).
4. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет. Простой пример выбора страницы видеопамяти с номером 1 показан в листинге 4.23.

Листинг 4.23. Использование функции 1013h

```
mov AX, 1013h; код функции 1013h
mov BL, 1    ; выбрать номер страницы
mov BH, 01h  ; страница с номером 1
int 10h      ; вызываем прерывание
```

4.2.27. Функция *1015h*

Функция позволяет получить информацию о цвете для указанного регистра ЦАП.

Использование:

1. В регистр AX следует поместить код функции 1015h.
2. В регистр BL надо записать номер регистра палитры ЦАП (от 00h до FFh).
3. Вызвать прерывание int 10h.

Выход:

После успешного завершения функции регистры будут содержать следующие данные: CH — зеленый, CL — синий, DH — красный. Пример использования функции 1015h представлен в листинге 4.24.

Листинг 4.24. Использование функции 1015h

```
mov AX, 1015h; код функции 1015h
mov BL, 03h  ; читаем регистр 4
int 10h      ; вызываем прерывание
mov GREEN, CH; читаем значение насыщенности зеленого цвета
mov BLUE, CL ; читаем значение насыщенности синего цвета
mov RED, DH  ; читаем значение насыщенности красного цвета
```

К сожалению, нет возможности проверить, правильно ли завершилась функция. Содержание регистра AX многими BIOS разрушается.

4.2.28. Функция *1017h*

Данная функция позволяет получить значения насыщенности цветов для целой группы регистров ЦАП.

Использование:

1. В регистр AX следует поместить код функции 1017h.
2. В регистр BX надо записать номер первого регистра ЦАП (от 00h до FFh) в группе.
3. В регистр CX требуется записать количество регистров в группе.
4. В регистр ES:DX следует поместить указатель на список описателей для каждого регистра группы. Размер каждого описателя равен трем байтам (красный, синий, зеленый). Значение каждого байта цвета лежит в промежутке от 0 до 63. Общий размер всей группы должен быть равен 3*CX байтов.
5. Вызвать прерывание int 10h.

Выход:

После выполнения функции выделенный список будет заполнен данными о насыщенности по каждому цвету.

4.2.29. Функция *101Ah*

Функция позволяет прочитать состояние страницы видеопамяти для регистров цвета ЦАП.

Использование:

1. В регистр AX следует поместить код функции 101Ah.
2. Вызвать прерывание int 10h.

Выход:

После выполнения функции регистр BL будет хранить число страниц (00h — 4, 01h — 16), а регистр BH — номер текущей страницы (00h—0Fh). Простой пример для получения информации о текущей странице показан в листинге 4.25.

Листинг 4.25. Использование функции 101Ah

```
mov AX, 101Ah ; код функции 101Ah
int 10h       ; вызываем прерывание
mov NPAGES, BL; количество страниц
mov CPAGE, BH ; номер текущей страницы
```

4.2.30. Функция *101Bh*

Функция позволяет выполнить полутоновое преобразование над группой регистров цвета.

Использование:

1. В регистр AX следует поместить код функции 101Bh.
2. В регистр BX нужно записать номер начального регистра (от 00h до FFH).
3. В регистр CX требуется записать количество регистров, подлежащих преобразованию.
4. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.31. Функция *1100h*

Эта функция позволяет в текстовом режиме загрузить пользовательские наборы символов, иначе говоря, шрифты.

Использование:

1. В регистр AX следует поместить код функции 1100h.
2. В регистр BH нужно записать число байтов для одного символа.
3. В регистр BL нужно записать номер блока знакогенератора. Знакогенератор представляет собой область памяти, в которой могут храниться до 8-ми наборов символов (в виде отдельных блоков). Как правило, в большинстве случаев применяется первый блок с номером 0.
4. В регистр CX следует поместить число символов.
5. В регистр DX следует поместить смещение к первому символу из набора.
6. В ES:BP необходимо записать указатель на пользовательский набор символов.
7. Вызвать прерывание int 10h.

Выход:

Возвращаемого значения нет.

4.2.32. Функция *1A00h*

Данная функция позволяет определить тип подключенного дисплея.

Использование:

1. В регистр AX следует поместить код функции 1A00h.
2. Вызвать прерывание int 10h.

Выход:

После успешного завершения функции регистры будут иметь следующий вид: в AL будет записано значение 1Ah, если функция поддерживается, в BL — код текущего дисплея (табл. 4.4), а в BH — дополнительный код дисплея (табл. 4.4). Данную функцию удобно использовать для проверки наличия VGA-совместимого дисплея, но она поддерживается далеко не всеми BIOS.

Таблица 4.4. *Код дисплея*

Код дисплея	Описание
00h	Дисплей отсутствует
01h	Монохромный адаптер и дисплей
02h	Адаптер CGA и цветной дисплей
04h	Адаптер EGA и цветной дисплей
05h	Адаптер EGA и монохромный дисплей

Таблица 4.4 (окончание)

Код дисплея	Описание
06h	Адаптер PGA и цветной дисплей
07h	Адаптер VGA и монохромный аналоговый дисплей
08h	Адаптер VGA и цветной аналоговый дисплей
0Ah	Адаптер MCGA и цветной цифровой дисплей
0Bh	Адаптер MCGA и монохромный аналоговый дисплей
0Ch	Адаптер MCGA и монохромный аналоговый дисплей
FFh	Неизвестный тип дисплея

Кроме рассмотренных возможностей BIOS, существует дополнительный набор функций VESA BIOS, поддерживающий современные режимы работы видеоадаптеров. Этот набор также использует прерывание под номером `int 10h`. Стандарт VBE (VESA BIOS Extension) позволяет управлять SVGA (Super VGA) видеоадаптерами, установленными на современных персональных компьютерах. Необходимость работы с VESA BIOS возникает при решении нестандартных задач: создание операционной системы, разработка драйверов для видеоплат, разработка другого специального оборудования, требующего вывода информации на цветной дисплей. Набор стандартных функций может быть дополнен собственными функциями разработчика оборудования.

Рассмотрим подробнее набор функций VESA BIOS. Каждая функция состоит из общего кода `4Fh` и номера подфункции (например, `00h`). После выполнения функции, если она поддерживается, в регистр `AL` заносится значение `4Fh`. Кроме того, регистр `AH` хранит код ошибки, по которому можно судить о результате операции. Любое значение кода ошибки, отличное от нуля, однозначно укажет на какой-либо сбой. Возможные значения кода ошибки приведены в табл. 4.5.

***Таблица 4.5.** Коды ошибок VESA BIOS*

Код ошибки	Описание
00h	Функция успешно завершена
01h	Функция завершена с ошибкой
02h	Функция не поддерживается установленным видеоконтроллером или дисплеем
03h	Функция не может быть выполнена в текущем режиме

Стандарт VESA BIOS поддерживает определенные режимы, перечисленные в табл. 4.6. Стандартные режимы начинаются с `100h`.

Таблица 4.6. *Список стандартных режимов VESA BIOS*

Код	Разрешение	Количество цветов	Тип
100h	640 × 400	256	Графический
101h	640 × 480	256	Графический
102h	800 × 600	16	Графический
103h	800 × 600	256	Графический
104h	1024 × 768	16	Графический
105h	1024 × 768	256	Графический
106h	1280 × 1024	16	Графический
107h	1280 × 1024	256	Графический
108h	80 × 60	16	Текстовый
109h	132 × 25	16	Текстовый
10Ah	132 × 43	16	Текстовый
10Bh	132 × 50	16	Текстовый
10Ch	132 × 60	16	Текстовый
10Dh	320 × 200	32K (1:5:5:5)	Графический
10Eh	320 × 200	64K (5:6:5)	Графический
10Fh	320 × 200	16,8M (8:8:8)	Графический
110h	640 × 480	32K (1:5:5:5)	Графический
111h	640 × 480	64K (5:6:5)	Графический
112h	640 × 480	16,8M (8:8:8)	Графический
113h	800 × 600	32K (1:5:5:5)	Графический
114h	800 × 600	64K (5:6:5)	Графический
115h	800 × 600	16,8M (8:8:8)	Графический
116h	1024 × 768	32K (1:5:5:5)	Графический
117h	1024 × 768	64K (5:6:5)	Графический
118h	1024 × 768	16,8M (8:8:8)	Графический
119h	1280 × 1024	32K (1:5:5:5)	Графический

Таблица 4.6 (окончание)

Код	Разрешение	Количество цветов	Тип
11Ah	1280 × 1024	64K (5:6:5)	Графический
11Bh	1280 × 1024	16,8M (8:8:8)	Графический
81FFh	Специальный режим для сохранения текущего состояния видеопамяти		

4.2.33. Функция *4F00h*

Функция позволяет получить информацию о видеоконтроллере: версию VESA BIOS, поддерживаемые возможности устройства, а также различную информацию о производителе видеоадаптера.

Использование:

1. В регистр `AX` следует поместить код функции `4F00h`.
2. В `ES:DI` необходимо поместить указатель на буфер размером 512 байт, куда будет записана вся полученная информация. Формат буфера представлен в табл. 4.7
3. Вызвать прерывание `int 10h`.

Выход:

Если функция не поддерживается, регистр `AL` будет содержать значение `4Fh`. Регистр `AH` будет установлен в соответствии с табл. 4.5. Выделенный буфер будет хранить всю полученную информацию о видеоконтроллере.

Таблица 4.7. Формат буфера для VESA BIOS

Смещение	Описание
00h	Сигнатура стандарта VBE VESA
04h	Версия VESA BIOS (0300h для третьей или 0200h для второй, 0102h для версии 1.2). Младший байт указывает на минорную версию, а старший — на мажорную
06h	Указатель на OEM-строку, которая может содержать описание микросхемы видеоадаптера или описание продукта для правильного использования драйвером. Длина строки не регламентируется, но рекомендуется ограничивать ее 256 байтами
0Ah	Описание поддерживаемых возможностей видеоадаптера (табл. 4.8)
0Eh	Указатель на список поддерживаемых видеорежимов (каждый режим занимает два байта и содержит все возможные режимы видеоадаптера, даже те, которые не поддерживает монитор)

Таблица 4.7 (окончание)

Смещение	Описание
12h	Полный размер видеопамяти (выражен числом блоков по 64 Кбайт). Например, для 32 Мбайт памяти будет возвращено значение 512 (512 * 64)
14h	Код программной версии VBE
16h	Указатель на строку с именем производителя
1Ah	Указатель на строку с названием устройства
1Eh	Указатель на строку с версией устройства
22h	Зарезервированная область VBE (222 байта)
100h	Область данных (256 байт)

***Таблица 4.8.** Поддерживаемые возможности видеоадаптера*

Бит	Описание
0	Разрядность ЦАП (1 — 8 бит, 0 — 6 бит)
1	Поддержка VGA (1 — не совместим, 0 — совместим)
2	Использование RAMDAC (1 — используется бит очистки функции 09h при работе с большими блоками данных, 0 — обычный режим работы)
3	Поддержка стереоскопического синхросигнала видеоадаптером (1 — поддерживается, 0 — не поддерживается)
4	Поддержка стереосигнала (1 — через разъем VESA EVC, через стереоразъем VESA)
5—31	Резерв

Рассмотрим простой пример кода для получения информации об установленном видеоконтроллере (листинг 4.26).

Листинг 4.26. Использование функции 4F00h

```
; выделяем память для хранения информации, размером 512 байт
VIDEO_INFO db 512 dup (?)
TOTAL_MEMORY dd ?
; код программы
mov BX, DS    ; адресуем сегмент данных
mov ES, BX    ; передаем адрес
mov DI, offset VIDEO_INFO
```

```
mov AX, 4F00h; код функции 4F00h
int 10h      ; вызываем прерывание
cmp AL, 4Fh  ; если не равно 4Fh
jne NO_SUP   ; функция не поддерживается
cmp AH, 0    ; если не 0
jnz ERROR_HND; произошла ошибка
; сохраняем полный размер видеопамяти в килобайтах
xor EAX, EAX ; обнуляем регистр
mox AX, [word ptr VIDEO_INFO + 12h]
shl AX, 6    ; умножаем результат на 64
; сохраняем значение в переменную TOTAL_MEMORY
mov TOTAL_MEMORY, EAX
```

4.2.34. Функция *4F01h*

Данная функция позволяет получить различную информацию для указанного режима.

Использование:

1. В регистр AX следует поместить код функции 4F01h.
2. В регистр CX нужно записать номер режима.
3. В ES:DI помещается указатель на буфер размером 256 байт, куда будет записана вся полученная информация. Формат буфера представлен в табл. 4.8.
4. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. Выделенный буфер будет хранить всю полученную информацию о видеоконтроллере. Формат буфера показан в табл. 4.9.

Таблица 4.9. *Формат описателя видеорежима*

Смещение	Версия VESA BIOS	Описание
00h	Для всех версий	Атрибуты режима (табл. 4.10)
02h		Поддержка атрибутов окна A (табл. 4.11)
03h		Поддержка атрибутов окна B (табл. 4.11)
04h		Поддержка минимального размера в килобайтах, необходимого для размещения окна в буфере видеопамяти

Таблица 4.9 (продолжение)

Смещение	Версия VESA BIOS	Описание
06h		Указывает на размер окна в килобайтах
08h		Адрес сегмента окна А в адресном пространстве процессора для реального режима
0Ah		Адрес сегмента окна В в адресном пространстве процессора для реального режима
0Ch		Указывает на смещение адреса функции, работающей с окнами в памяти
10h		Указывает на полное число байтов для одной строки растра (развертки)
12h	Для версии 1.2 и старше	Определяет горизонтальное разрешение дисплея в пикселах или символах
14h		Определяет вертикальное разрешение дисплея в пикселах или символах
16h		Определяет ширину символьной ячейки в пикселах
17h		Определяет высоту символьной ячейки в пикселах
18h		Определяет количество плоскостей в памяти, доступных указанному режиму
19h		Указывает на количество битов, выделенных указанным режимом на один пиксел
1Ah		Указывает на количество банков памяти для развертки
1Bh		Указывает на тип организации общей памяти (табл. 4.12)
1Ch		Указывает на размер одного банка памяти развертки, выраженный в однокилобайтных модулях
1Dh		Указывает на количество (минус 1) полных копий экрана, которые могут быть размещены в буфере памяти
1Eh		Зарезервировано и равно 1 для версии 3.0
1Fh	Для Direct Color	Размер в битах красной составляющей цвета пиксела
20h		Указывает на битовую позицию красной составляющей цвета пиксела

Таблица 4.9 (продолжение)

Смещение	Версия VESA BIOS	Описание
21h		Размер в битах зеленой составляющей цвета пиксела
22h		Указывает на битовую позицию зеленой составляющей цвета пиксела
23h		Размер в битах синей составляющей цвета пиксела
24h		Указывает на битовую позицию синей составляющей цвета пиксела
25h		Размер в битах резервной составляющей цвета пиксела
26h		Указывает на битовую позицию резервной составляющей цвета пиксела
27h		Атрибуты режима Direct Color (табл. 4.13)
28h	Для версии 2.0 и старше	Физический 32-разрядный адрес буфера памяти
2Ch		Зарезервировано и должно быть равно 0
30h		Зарезервировано и должно быть равно 0
32h	Для версии 3.0 и старше	Указывает на количество байтов в одной строке растра (развертки) для режима линейной адресации
34h		Указывает на количество (минус 1) полных копий экрана, которые могут быть размещены в буфере памяти для страничного режима
35h		Указывает на количество (минус 1) полных копий экрана, которые могут быть размещены в буфере памяти для режима линейной адресации
36h		Размер в битах красной составляющей цвета пиксела для режима линейной адресации
37h		Указывает на битовую позицию красной составляющей цвета пиксела для режима линейной адресации
38h		Размер в битах зеленой составляющей цвета пиксела для режима линейной адресации
39h		Указывает на битовую позицию зеленой составляющей цвета пиксела для режима линейной адресации

Таблица 4.9 (окончание)

Смещение	Версия VESA BIOS	Описание
40h		Размер в битах синей составляющей цвета пиксела для режима линейной адресации
41h		Указывает на битовую позицию синей составляющей цвета пиксела для режима линейной адресации
42h		Размер в битах резервной составляющей цвета пиксела для режима линейной адресации
43h		Указывает на битовую позицию резервной составляющей цвета пиксела для режима линейной адресации
44h		Указывает на максимальную частоту обновления пиксела на дисплее, измеряемую в герцах
48h		Зарезервированный остаток 189 байт

Таблица 4.10. *Атрибуты режима*

Бит	Описание
0	Поддержка режима текущей аппаратной конфигурацией (1 — поддерживается, 0 — не поддерживается)
1	Зарезервирован и всегда равен 1
2	Поддержка BIOS функций TTY (1 — поддерживает, 0 — не поддерживает) для вывода текста
3	Режим цветности (1 — цветной, 0 — монохромный)
4	Тип режима (1 — графический, 0 — текстовый)
5	Поддержка VGA (1 — не поддерживается, 0 — поддерживается)
6	Поддержка оконной адресации памяти (1 — не поддерживается, 0 — поддерживается)
7	Поддержка линейной адресации памяти (1 — поддерживается, 0 — не поддерживается)
8	Поддержка режима двойного сканирования (1 — поддерживается, 0 — не поддерживается)
9	Поддержка режима чередования (1 — поддерживается, 0 — не поддерживается)

Таблица 4.10 (окончание)

Бит	Описание
10	Поддержка аппаратной тройной буферизации (1 — поддерживается, 0 — не поддерживается)
11	Поддержка аппаратной стереоскопии дисплея (1 — поддерживается, 0 — не поддерживается)
12	Поддержка сдвоенного начального адреса (1 — поддерживается, 0 — не поддерживается)
13—15	Резерв

Таблица 4.11. *Атрибуты окна*

Бит	Описание
0	Поддержка перемещения окна или окон (1 — поддерживается, 0 — не поддерживается)
1	Поддержка чтения из окна (1 — поддерживается, 0 — не поддерживается)
2	Поддержка записи в окно (1 — поддерживается, 0 — не поддерживается)
3—7	Резерв

Таблица 4.12. *Тип организации памяти*

Код	Описание
00h	Текстовый режим
01h	Графический режим CGA
02h	Графический режим Hercules
03h	Плоский режим
04h	Режим с упакованными пикселами
05h	Не режим цепочки под номером 4 и 256 цветов
06h	Режим Direct Color
07h	YUV-режим
08h—0Fh	Резерв
10h—FFh	Резерв

Таблица 4.13. Атрибуты режима Direct Color

Бит	Описание
0	Состояние цветового шаблона ЦАП (1 — программируемый, 0 — фиксированный)
1	Состояние битов резервных полей цвета (1 — могут использоваться в программе, 0 — зарезервированы)

4.2.35. Функция *4F02h*

Функция позволяет установить поддерживаемый VESA BIOS режим.

Использование:

1. В регистр AX следует поместить код функции 4F02h.
2. В регистр BX нужно записать номер видеорежима (см. табл. 4.6). Кроме значения режима можно определить дополнительные параметры (табл. 4.14).
3. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5.

Таблица 4.14. Формат регистра BX для функции 4F02h

Бит	Описание
0—8	Номер режима (см. табл. 4.6)
9—10	Зарезервированы и должны быть установлены в 0
11	Установка частоты обновления (1 — использовать установленную пользователем, 0 — использовать заданную по умолчанию)
12—13	Зарезервированы и должны быть установлены в 0
14	Режим буфера памяти (1 — использовать линейную адресацию буфера видеопамяти, 0 — использовать оконную (постраничную) модель буфера памяти)
15	Установка очистки памяти (1 — не очищать память, 0 — очищать память дисплея)

Простой пример для установки режима под номером 102h (800 × 600 × 16) показан в листинге 4.27.

Листинг 4.27. Использование функции 4F02h

```
mov AX, 4F02h; код функции 4F02h
mov BX, 102h ; номер режима
int 10h      ; вызываем прерывание
cmp AL, 4Fh  ; если не равно 4Fh
jne NO_SUP   ; функция не поддерживается
cmp AH, 0    ; если не 0
jnz ERROR_HND; произошла ошибка
```

4.2.36. Функция *4F03h*

Эта функция позволяет узнать номер текущего видеорежима. При этом правильное определение режима гарантируется только тогда, когда для установки режима использовалась функция 4F02h.

Использование:

1. В регистр AX следует поместить код функции 4F03h.
2. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения в регистр BX будет записан номер текущего видеорежима. Формат этого значения показан в табл. 4.15.

Таблица 4.15. *Формат регистра BX для функции 4F03h*

Бит	Описание
0—13	Номер режима (см. табл. 4.6)
14	Режим буфера памяти (1 — используется линейная адресация буфера видеопамяти, 0 — используется оконная (постраничная) модель буфера памяти)
15	Очистка памяти (1 — память не очищается, 0 — память очищается)

Рассмотрим простой пример для установки режима под номером 102h (800 × 600 × 16) и получения текущего номера видеорежима (листинг 4.28).

Листинг 4.28. Использование функции 4F03h

```
mov AX, 4F02h; код функции 4F02h
mov BX, 102h ; номер режима
```

```
int 10h       ; вызываем прерывание
cmp AL, 4Fh   ; если не равно 4Fh
jne NO_SUPPRT; функция не поддерживается
cmp AH, 0     ; если не 0
jnz ERROR_HND; произошла ошибка
mov AX, 4F03h; код функции 4F03h
xor BX, BX    ; обнуляем регистр
int 10h       ; вызываем прерывание
and BX, 3FFFh; выделяем номер режима
cmp BX, 102h ; проверяем режим
jne ERROR_HND; номер режима не соответствует установленному
```

4.2.37. Функция *4F04h*

Данная функция позволяет сохранить или восстановить состояние видеоадаптера дисплея. Функция сохраняет все параметры видеоадаптера для текущего режима и может восстановить их, если режим был изменен.

Использование:

1. В регистр AX следует поместить код функции 4F04h.
2. В регистр DL следует записать один из следующих кодов:
   * 00h — определить размер буфера для сохранения данных;
   * 01h — сохранить текущее состояние;
   * 02h — восстановить состояние.
3. В регистр CX нужно записать слово (16 бит), описывающее дополнительные указания. Формат слова приведен в табл. 4.16.
4. В ES:BX надо поместить указатель на буфер данных. Если в регистре DL указан код 00h, определять буфер не требуется.
5. Вызвать прерывание int 10h.

Таблица 4.16. *Формат регистра* CX *для функции* 4F04h

Бит	Описание
0	Состояние аппаратных средств видеоадаптера
1	Состояние данных в BIOS
2	Состояние ЦАП
3	Состояние регистров видеоадаптера
4—15	Не используются и должны быть установлены в 0

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения регистр BX будет хранить необходимое количество блоков (по 64 байта каждый) для буфера данных. Естественно, это касается только операции определения размера буфера (в DL находится 00h).

4.2.38. Функция *4F05h*

Функция позволяет управлять окнами памяти на экране дисплея. Перед использованием данной функции следует убедиться, что текущий режим поддерживает оконную адресацию памяти, а также координаты окна и минимальный размер, выделенный для окна. Все эти данные можно получить с помощью функции 4F01h.

Использование:

1. В регистр AX следует поместить код функции 4F05h.
2. В регистр BH следует записать один из следующих кодов: 00h — установить окно в памяти, 01h — получить текущее окно в памяти.
3. В регистр BL нужно записать номер окна: 00h — окно A, 01h — окно B.
4. В регистр DX надо поместить номер окна в видеопамяти в единицах детализации окна (например, килобайтах). Используется только для операции установки окна в памяти (00h).
5. Для 32-разрядного защищенного режима дополнительно необходимо установить селектор для регистров в отображаемой памяти.
6. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения регистр DX будет хранить номер окна в единицах детализации окна. Это истинно только для операции получения текущего окна в памяти (01h).

4.2.39. Функция *4F06h*

Эта функция предназначена для получения или установки логической длины строки развертки. Данные, получаемые от функции, позволяют установить логический буфер памяти для дисплея, который всегда больше отображаемой на экране области. Функция может с успехом применяться не только в графических, но и в текстовых режимах.

Использование:

1. В регистр AX следует поместить код функции 4F06h.
2. В регистр BL нужно записать код операции:
   - 00h — установить длину строки развертки в пикселах;
   - 01h — получить длину строки развертки;
   - 02h — установить длину строки развертки в байтах;
   - 03h — получить максимально возможную длину строки развертки.
3. В регистр CX надо записать определенное значение, в зависимости от кода операции. Для операции 00h нужно указать желательную ширину в пикселах, а для 02h — желательную ширину в байтах. В остальных случаях (01h и 03h) этот регистр не используется.
4. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения регистр BX будет хранить количество байт в одной строке развертки, а регистр CX — фактическое число пикселов для одной строки развертки, округленное до целого значения.

4.2.40. Функция *4F07h*

Функция позволяет получить или установить начальные координаты экрана. Для 32-разрядного режима в регистр CX следует записать начальный адрес дисплея (биты 0—15), а в регистр DX — вторую часть адреса дисплея (биты 16—31).

Использование:

1. В регистр AX следует поместить код функции 4F07h.
2. В регистр BH нужно записать нулевое значение.
3. В регистр BL нужно записать код операции. Возможные значения показаны в табл. 4.17.
4. В регистр ECX следует записать следующее значение: для кодов операции 02h и 82h надо установить начальный адрес дисплея в байтах; для кодов 03h и 83h — левый начальный адрес экрана в байтах.
5. В регистр EDX следует записать следующее значение: для кодов операции 03h и 83h — правый начальный адрес экрана в байтах.
6. Если код операции равен 00h или 80h, в регистр CX записывается первый отображаемый пиксел в строке развертки.

7. Если код операции равен 00h или 80h, в регистр DX записывается первая отображаемая строка развертки.
8. Только для 32-разрядного защищенного режима надо указать селектор для регистров в отображаемой памяти.
9. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения регистр BH будет хранить значение 0, если код операции равен 01h. В регистр CX будет помещен первый отображаемый пиксел в строке развертки (код операции равен 01h) или 0 (код операции равен 04h). В регистр CX будет помещена первая отображаемая строка развертки, если код операции равен 01h.

Таблица 4.17. *Коды операций для функции 4F07h*

Код операции	Описание
	16-разрядный режим
00h	Установить начальные координаты экрана
01h	Получить начальные координаты экрана
02h	Установить список начальных координат экрана
03h	Установить список начальных координат для стереоскопического экрана
04h	Получить список начальных координат экрана
05h	Включить стереоскопический режим
06h	Выключить стереоскопический режим
80h	Установить начальные координаты экрана в течение обратного хода луча по вертикали
82h	Установить начальные координаты экрана в течение обратного хода луча по вертикали (альтернативный вариант)
83h	Установить начальные координаты экрана в течение обратного хода луча по вертикали для стереоскопического экрана
	32-разрядный режим
00h	Установить начальные координаты экрана
80h	Установить начальные координаты экрана в течение обратного хода луча по вертикали

4.2.41. Функция *4F08h*

Эта функция позволяет получить и установить формат регистров палитры для ЦАП видеоконтроллера.

Использование:

1. В регистр AX следует поместить код функции 4F08h.
2. В регистр BL следует записать один из следующих кодов: 00h — установить формат палитры, 01h — получить текущий формат палитры.
3. В регистр BH нужно записать первичный набор битов цвета. Используется только во время установки формата палитры (00h).
4. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения регистр BH будет хранить текущее количество битов для цвета.

4.2.42. Функция *4F09h*

Функция позволяет получить или установить данные для регистров палитры.

Использование:

1. В регистр AX следует поместить код функции 4F09h.
2. В регистр BL следует записать один из следующих кодов:
   - 00h — установить данные для палитры;
   - 01h — получить текущие данные палитры;
   - 02h — установить данные для второй палитры;
   - 03h — получить текущие данные для второй палитры;
   - 80h — установить данные для палитры в течение обратного хода луча.
3. В регистр CX нужно записать количество обновляемых регистров палитры (максимум 256).
4. В регистр DX требуется записать номер первого регистра палитры, с которого начнется обновление данных.
5. В ES:DI нужно поместить указатель на таблицу значений палитры.
6. Для 32-разрядного режима следует в DS указать селектор для регистров в отображаемой памяти.
7. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5.

4.2.43. Функция *4F0Ah*

Данная функция позволяет получить интерфейс защищенного режима. Функция возвращает указатель на таблицу кодов для обычного защищенного режима. Функция является необязательной для VBE версии 3.0 и может не поддерживаться видеоадаптером.

Использование:

1. В регистр AX следует поместить код функции 4F0Ah.
2. В регистр BL следует записать значение 00h.
3. Вызвать прерывание int 10h.

Выход:

Если функция не поддерживается, регистр AL будет содержать значение 4Fh. Регистр AH будет установлен в соответствии с табл. 4.5. В случае успешного выполнения в ES будет записан сегмент таблицы реального режима, а в DI — смещение к таблице. В регистр CX запишется значение длины таблицы (включая защищенный режим) в байтах.

На этом можно завершить рассмотрение возможностей BIOS по управлению видеоадаптером и познакомиться с непосредственным программированием аппаратных портов видеоконтроллера.

4.3. Использование портов

Перед тем как рассказать о программировании портов видеоконтроллера, хочу сделать важное замечание: будьте предельно ВНИМАТЕЛЬНЫ и ОСТОРОЖНЫ. Случайно сделанная ошибка (возможны описки и в самой книге) может привести к выходу из строя дисплея вплоть до его возгорания. Перепроверяйте по несколько раз свои действия, а лучше советуйтесь со знающими специалистами. Для программирования портов видеоадаптера рекомендую использовать старый монитор, с которым не жалко будет расстаться. Несоблюдение этих простых правил может повлечь серьезные последствия, за которые читатель будет нести ответственность самостоятельно.

Стандартный VGA-совместимый видеоконтроллер поддерживает следующие типы регистров:

1. Внешние регистры.
2. Регистры графического контроллера.
3. Регистры контроллера атрибутов.

4. Регистры контроллера CRT.
5. Регистры ЦАП.
6. Регистры синхронизатора.

Список всех адресов регистров для стандартного видеоконтроллера приведен в табл. 4.18. Рассмотрим работу с регистрами подробнее.

Таблица 4.18. *Адреса регистров видеоконтроллера*

Адрес регистра	Имя регистра
3B4h	CRTC Controller Address Register
3B5h	CRTC Controller Data Register
3BAh	Input Status #1 Register (чтение) или Feature Control Register (запись)
3C0h	Attribute Address/Data Register
3C1h	Attribute Data Read Register
3C2h	Input Status #0 Register (чтение) или Miscellaneous Output Register (запись)
3C4h	Sequencer Address Register
3C5h	Sequencer Data Register
3C7h	DAC State Register (чтение) или DAC Address Read Mode Register (запись)
3C8h	DAC Address Write Mode Register
3C9h	DAC Data Register
3CAh	Feature Control Register (чтение)
3CCh	Miscellaneous Output Register (чтение)
3CEh	Graphics Controller Address Register
3CFh	Graphics Controller Data Register
3D4h	CRTC Controller Address Register
3D5h	CRTC Controller Data Register
3DAh	Input Status #1 Register (чтение) или Feature Control Register (запись)

4.3.1. Внешние регистры

Данные регистры позволяют управлять различными общими настройками, а также получать различную информацию о состоянии контроллера.

В данную группу входят четыре основных регистра:

- регистр общих настроек (Miscellaneous Output Register). Доступ для записи осуществляется через порт 3C2h, для чтения — через порт 3CCh;
- регистр особенностей (Feature Control Register). Доступ для записи производится через порт 3DAh (цветной режим) и 3BAh (монохромный режим), для чтения — через порт 3CAh;
- первый регистр состояния (Input Status #0 Register). Доступен только для чтения через порт 3C2h;
- второй регистр состояния (Input Status #1 Register). Доступен только для чтения через порт 3DAh (цветной режим) или 3BAh (монохромный режим).

4.3.1.1. Регистр общих настроек

Размер регистра равен 8 бит. Регистр управляет частотой синхронизации, выбором страниц памяти, полярностью горизонтальных и вертикальных импульсов. Формат регистра представлен в табл. 4.19.

Таблица 4.19. *Формат регистра общих настроек*

Бит	Описание
0	Выбор адресов ввода-вывода для контроллера CRT
1	Доступ к видеопамяти
2—3	Частота синхронизации
4	Резерв
5	Выбор четной или нечетной страницы видеопамяти
6	Полярность горизонтальной синхронизации
7	Полярность вертикальной синхронизации

Остановимся подробнее на каждой строке таблицы.

- Бит 0 используется для установки адресов управления контроллера CRT (катодно-лучевой трубки). Если бит установлен в 1, используются порты 3D4h и 3D5h. Если бит установлен в 0, будут выбраны порты 3B4h и 3B5h.
- Бит 1 управляет доступом к видеопамяти устройства. Установка бита в 1 разрешает системе доступ к памяти видеоконтроллера. Установка бита в 0 запрещает доступ.
- Биты 2 и 3 управляют выбором частоты синхронизации. Значение 00h указывает на выбор частоты 25 МГц (точнее 25,175), а значение 01h — на частоту 28 МГц (или 28,322).

- Бит 4 зарезервирован и не используется.
- Бит 5 управляет выбором четной (значение равно 0) или нечетной (значение равно 1) страницы видеопамяти.
- Бит 6 управляет полярностью горизонтальной синхронизации. Значение 0 соответствует положительной полярности сигнала.
- Бит 7 управляет полярностью вертикальной синхронизации. Значение 0 соответствует положительной полярности сигнала. Может использоваться совместно с битом 6 для управления вертикальным размером экрана.

4.3.1.2. Регистр особенностей

Размер регистра равен 8 бит. Реально задействованы только 0 и 1 биты, и оба являются служебными. Практическое применение этого регистра не документируется.

4.3.1.3. Первый регистр состояния

Размер регистра равен 8 бит. Бит 7 отвечает за прерывание обратного хода луча. Биты 5 и 6 указывают на присутствие дополнительных устройств. Бит 4 определяет наличие монитора. Данный регистр может иметь и другой формат, поэтому на него не стоит полагаться.

4.3.1.4. Второй регистр состояния

Размер регистра равен 8 бит. Он позволяет получить информацию об обратном ходе луча. Формат регистра представлен в табл. 4.20.

Таблица 4.20. Формат второго регистра состояния

Бит	Описание
0	Обратный ход луча по горизонтали или вертикали
1—2	Резерв
3	Обратный ход луча по вертикали
4—7	Резерв

При этом бит 0 указывает на наличие обратного хода луча по горизонтали или вертикали. Значение 0 определяет обратный ход луча.

Бит 3 указывает на наличие обратного хода луча только по вертикали. Если значение равно 1, происходит обратный ход луча.

Рассмотрим примеры работы с внешними регистрами. Попробуем установить частоту синхронизации 25 МГц (листинг 4.29).

Листинг 4.29. Установка новой частоты синхронизации

```
xor AX, AX    ; обнуляем регистр
mov DX, 03CCh ; выбираем порт для чтения
in  AL, DX    ; читаем байт из порта
and AL, 01100b; устанавливаем частоту 25 МГц
mov DX, 03C2h ; выбираем порт для записи
out DX, AX    ; записываем значение в порт
```

Аналогичный пример для C++ показан в листинге 4.30.

Листинг 4.30. Установка новой частоты синхронизации в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
// читаем байт из порта 03CCh
inPort ( 0x03CC, &dwResult, 1);
// устанавливаем частоту 25 МГц
dwResult &= 0x0C;
// записываем значение в порт 03C2h
outPort (0x03C2, dwResult, 1);
```

Рассмотрим еще один пример для определения обратного хода луча по вертикали (листинг 4.31).

Листинг 4.31. Определение момента обратного хода луча по вертикали

```
mov DX, 03DAh ; выбираем порт для чтения
@End_Beam
in  AL, DX    ; читаем байт из порта
test AL, 8    ; состояние бита 3
jnz @End_Beam ; ждем окончание обратного хода луча
@Start_Beam:  ; ждем начала следующего луча
in  AL, DX    ; читаем байт из порта
and AL, 01000b; проверяем бит 3
jz  @Start_Beam
```

Аналогичный пример для C++ будет представлен в листинге 4.32.

Листинг 4.32. Определение момента обратного хода луча по вертикали в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
// читаем байт из порта 03DAh
int iTimeWait = 1000;
```

```
// ждем окончание обратного хода луча
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x03DA, &dwResult, 1);
      if ( (dwResult & 0x08) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
int iTimeWait = 1000;
// ждем начала следующего луча
while ( -- iTimeWait > 0)
{
      // читаем состояние порта
      inPort ( 0x03DA, &dwResult, 1);
      if ( (dwResult & 0x08) == 0x00) break;
      // закончилось время ожидания
      if ( iTimeWait < 1) return MY_ERROR_TIME;
}
```

4.3.2. Регистры графического контроллера

Данные регистры управляют доступом процессора к видеопамяти. Для указания адреса используется порт 3CEh, а для передачи данных — порт 3CFh. Графический контроллер поддерживает 9 индексных регистров (от 0h до 8h). Каждый индексный регистр отвечает за определенные настройки. Список индексных регистров приведен в табл. 4.21.

Таблица 4.21. *Список индексных регистров графического контроллера*

Код индексного регистра	Описание
00h	Регистр установки/сброса
01h	Регистр разрешения для установки/сброса
02h	Регистр сравнения цвета
03h	Регистр управления циклическим сдвигом данных
04h	Регистр выбора плоскости чтения
05h	Регистр управления режимом работы
06h	Графический регистр общего назначения
07h	Регистр сброса цветовых плоскостей
08h	Регистр битовой маски

Принцип работы с графическим контроллером следующий. В порт адреса 3CEh следует записать код индексного регистра. После этого можно передавать или получать данные через порт данных 3CFh. Для выбора другого индексного регистра необходимо опять записать его код в порт 3CEh.

Далее рассмотрим индексные регистры подробнее.

4.3.2.1. Регистр *00h*

Размер регистра равен 8 бит. Данный регистр управляет цветовыми плоскостями, расположенными в видеопамяти. Биты с 0 по 3 представляют собой четыре плоскости. Эти биты используются режимами записи *(см. разд. 4.3.2.6)* под номерами 0 и 3.

4.3.2.2. Регистр *01h*

Размер регистра равен 8 бит. Этот регистр позволяет включать или отключать использование регистра 00h при работе с выбранной плоскостью. Биты с 0 по 3 представляют собой четыре плоскости в видеопамяти. Они используются в нулевом режиме записи и позволяют выбирать тип операции: получать ли данные для каждой плоскости от системы (процессора) или от установки соответствующего бита в индексном регистре 00h.

4.3.2.3. Регистр *02h*

Размер регистра равен 8 бит. Данный режим сравнивает выбранный цвет в регистре с цветом в памяти и возвращает результат сравнения. Биты с 0 по 3 представляют собой четыре плоскости, расположенные в видеопамяти. Процесс сравнения начинается с чтения (только в режиме чтения 1) значения цвета, после чего считывается значение цвета из памяти. В результате, если бит в памяти и бит в регистре совпадают, результирующий бит равен 1, иначе 0.

4.3.2.4. Регистр *03h*

Размер регистра равен 8 бит. Он предназначен для выбора различных логических операций, применяемых к передаваемым от процессора данным, а также позволяет выполнять циклический сдвиг вправо передаваемых данных. Логическая операция выполняется над данными и содержимым регистра-защелки. Формат регистра приведен в табл. 4.22.

Приведем некоторые пояснения к таблице.

- Биты 0—2 определяют целое значение, на которое будет выполнена логическая операция сдвига вправо. Используется для режимов записи 0 и 3.

- Биты 3 и 4 указывают на тип логической операции и могут принимать одно из следующих значений:
  - 00h — данные передаются без изменений;
  - 01h — данные преобразуются с помощью операции AND;
  - 02h — данные преобразуются с помощью операции OR;
  - 03h — данные преобразуются с помощью операции XOR. Данное поле применяется для режимов записи под номерами 0 и 2.

Таблица 4.22. *Формат индексного регистра* 03h

Бит	Описание
0—2	Индекс сдвига
3—4	Тип логической операции
4—7	Резерв

4.3.2.5. Регистр *04h*

Размер регистра равен 8 бит. Предназначен для выбора плоскости в видеопамяти, которая будет использована в режиме чтения (номер 0) для получения данных. Задействованы только 0 и 1 биты, с помощью которых выбирается номер плоскости (0—3).

4.3.2.6. Регистр *05h*

Размер регистра равен 8 бит. Он предназначен для установки различных режимов работы. Формат регистра представлен в табл. 4.23.

Таблица 4.23. *Формат индексного регистра* 05h

Бит	Описание
0—1	Номер режима записи
2	Резерв
3	Номер режима чтения
4	Адресация плоскостей
5	Режим чередования
6	Режим цвета
7	Резерв

Приведем следующие пояснения к таблице.

- Биты 0 и 1 позволяют выбрать режим записи. Существует четыре режима записи (от 0 до 3):
  - режим 0. Данные сначала сдвигаются в соответствии со значением индекса сдвига (регистр 03h), далее выполняется операция установки/сброса (регистр 00h), а затем указанная логическая операция (регистр 03h). После этого данные помещаются в регистр-защелку, и там с помощью битовой маски (регистр 08h) выбираются нужные биты, а затем окончательные битовые слои записываются в видеопамять в зависимости от установки индексного регистра 02h синхронизатора;
  - режим 1. Данные передаются непосредственно из 32-битного регистра-защелки в видеопамять и зависят только от установки индексного регистра 02h синхронизатора;
  - режим 2. Биты 0—3 записываются в соответствующие плоскости видеопамяти. После этого в регистре-защелке выполняется указанная логическая операция, и данные обрабатываются с учетом битовой маски (регистр 08h), а затем окончательные битовые слои записываются в видеопамять в зависимости от установки индексного регистра 02h синхронизатора;
  - режим 3. Данные обрабатываются, как если бы биты 3—0 регистра 00h были установлены в 1 (1111b). После этого выполняется операция сдвига в соответствии со значением индекса сдвига (регистр 03h), а затем используется логическая операция AND. Далее данные обрабатываются с учетом битовой маски (регистр 08h), а затем окончательные битовые слои записываются в видеопамять в зависимости от установки индексного регистра 02h синхронизатора.
- Бит 3 определяет один из возможных режимов чтения: 0 или 1. В режиме чтения 0 байт считывается с одной из четырех плоскостей (0—3). Номер плоскости выбирается в регистре 04h. В режиме чтения 1 выполняется сравнение цвета в видеопамяти с цветом, указанным в регистре 02h. Битовые плоскости, игнорируемые в регистре 07h, не сравниваются. В результате возвращается бит сравнения между цветом текущим и устанавливаемым.
- Бит 4, установленный в 1, позволит выбирать нечетные адреса плоскостей (1 или 3) для передаваемых от процессора данных, использующих нечетные адреса.
- Бит 5, установленный в 1, позволит регистрам графического контроллера обрабатывать поток данных последовательно-параллельным способом: нечетные биты записываются в нечетные плоскости или четные биты записываются в четные битовые плоскости.

- Бит 6, установленный в 1, позволит включить поддержку 256-цветного режима. Установка бита в 0 позволит использовать только 16 цветов.

4.3.2.7. Регистр *06h*

Размер регистра равен 8 бит. Он предназначен для установки различных параметров. Формат регистра представлен в табл. 4.24.

Таблица 4.24. *Формат индексного регистра* 06h

Бит	Описание
0	Управление режимом
1	Управление четностью
2—3	Выбор диапазона видеопамяти

Приведем некоторые пояснения к таблице.

- Бит 0, установленный в 1, включает графический режим, а запись 0 переводит видеоадаптер в текстовый режим.
- Бит 1, установленный в 1, позволит управлять четностью. При этом нечетные адреса будут записаны в нечетную плоскость, а четные — в четную плоскость памяти.
- Биты 2—3 позволяют установить доступный видеоадаптеру диапазон памяти. Поддерживаются следующие значения: 00h — A0000h—BFFFFh (128 Кбайт), 01h — A0000h—AFFFFh (64 Кбайт), 02h — B0000h—B7FFFh (32 Кбайт) и 03h — B8000h—BFFFFh (32 Кбайт).

4.3.2.8. Регистр *07h*

Размер регистра равен 8 бит. Он позволяет игнорировать указанные цветовые плоскости. Для выбора нужной плоскости используются биты 0—3. Данный регистр используется для режима чтения 1. Установка любого бита указывает на то, что соответствующая плоскость будет рассмотрена при сравнении. Установка бита в 0 позволяет режиму чтения игнорировать ее.

4.3.2.9. Регистр *08h*

Размер регистра равен 8 бит. Данный регистр (все 8 бит) используется в режимах записи 0, 2 и 3. Он позволяет включить или отключить возможность изменения одного или всех битов в байте адресуемых данных. Значение регистра влияет на все четыре плоскости. Если бит в регистре установлен в 1, значит, будет выбрано предыдущее значение бита. Если бит равен 0, будет использовано значение соответствующего бита из регистра-защелки.

Рассмотрим примеры работы с регистрами графического контроллера. Сначала получим значение диапазона памяти, используемой видеоконтроллером (листинг 4.33).

Листинг 4.33. Определение адреса выделенной памяти

```
xor AL, AL    ; обнуляем регистр
mov AL, 06h   ; записываем значение регистра 06h
mov DX, 03CEh ; выбираем порт для записи
out DX, AL    ; записываем значение в порт
mov DX, 03CFh ; выбираем порт для чтения
in  AL, DX    ; читаем байт из порта
and AL, 01100b; выделяем нужное значение
cmp AL, 0     ; если A0000h-BFFFFh (128 Кбайт)
je  MEM_128
cmp AL, 1     ; если A0000h-AFFFFh (64 Кбайта)
je  MEM_64
cmp AL, 2     ; если B0000h-B7FFFh (32 Кбайта)
je  MEM_32
cmp AL, 3     ; если B8000h-BFFFFh (32 Кбайта)
je  MEM_32_2
```

Аналогичный пример для C++ будет выглядеть так, как показано в листинге 4.34.

Листинг 4.34. Определение адреса выделенной памяти в C++

```
DWORD dwResult  = 0; // переменная для хранения результата
// записываем значение регистра 06h в порт 03CEh
outPort (0x03CE, 0x06, 1);
// читаем байт из порта 03CFh
inPort ( 0x03CF, &dwResult, 1);
// выделяем биты 2 и 3
dwResult &= 0x0C;
// сравниваем результат
switch ( dwResult)
{
    case 0: // A0000h-BFFFFh (128 Кбайт)
     break;
    case 1: // A0000h-AFFFFh (64 Кбайта)
     break;
    case 2: // B0000h-B7FFFh (32 Кбайта)
     break;
```

```
    case 3: // B8000h-BFFFFh (32 Кбайта)
     break;
}
```

Рассмотрим еще один пример, использующий регистр 04h для выбора плоскости под номером 1 (листинг 4.35).

Листинг 4.35. Выбор плоскости в видеопамяти

```
xor AL, AL   ; обнуляем регистр
mov AL, 04h  ; записываем значение регистра 04h
mov DX, 03CEh; выбираем порт для записи
out DX, AL   ; записываем значение в порт
mov DX, 03CFh; выбираем порт для записи
mov AL, 01b  ; плоскость номер 1
out DX, AL   ; записываем значение в порт
```

Аналогичный пример для C++ будет выглядеть так, как показано в листинге 4.36.

Листинг 4.36. Выбор плоскости в видеопамяти для C++

```
DWORD dwResult  = 0; // переменная для хранения результата
// записываем значение регистра 04h в порт 03CEh
outPort (0x03CE, 0x04, 1);
// записываем плоскость номер 1 в порт 03CFh
outPort (0x03CF, 0x01, 1);
```

Приведу еще один пример использования регистров 00h и 01h. Напишем код, который допускает использование только плоскости с номером 2 (листинг 4.37).

Листинг 4.37. Фиксация плоскости в видеопамяти

```
xor AL, AL   ; обнуляем регистр
mov AL, 00h  ; записываем значение регистра 00h
mov DX, 03CEh; выбираем порт для записи
out DX, AL   ; записываем байт в порт
mov DX, 03CFh; выбираем порт для записи
mov AL, 00h  ; обнуляем все 4 плоскости
out DX, AL   ; записываем байт в порт
mov AL, 01h  ; записываем значение регистра 01h
mov DX, 03CEh; выбираем порт для записи
out DX, AL   ; записываем байт в порт
```

```
mov DX, 03CFh; выбираем порт для записи
mov AL, 0100h; включаем плоскость номер 2
out DX, AL   ; записываем байт в порт
```

Поскольку значения портов 3CEh и 3CFh отличаются на одну единицу, можно использовать (для ассемблера) команды inc и dec. Например, чтобы передать данные в регистр 07h, сделайте так, как показано в листинге 4.38.

Листинг 4.38. Передача видеоданных

```
mov AL, 07h  ; записываем значение регистра 07h
mov DX, 03CEh; выбираем порт 03CEh
out DX, AL   ; записываем байт в порт
inc DX       ; выбираем соседний порт 03CFh
in  AL, DX   ; читаем байт из порта
dec DX       ; выбираем опять порт 03CEh
mov AL, 07h  ; записываем значение регистра 07h
out DX, AL   ; записываем байт в порт
inc DX       ; выбираем соседний порт 03CFh
in  AL, DX   ; читаем байт из порта
```

Приведенный пример ничего не делает, кроме демонстрации удобного способа выбора смежных портов. Далее в этой главе будет использоваться код с явным указанием всех портов, что делается исключительно для наглядности материала, но в своих программах читатель вправе выбирать наиболее удобный для себя вариант.

4.3.3. Регистры контроллера атрибутов

Данные регистры управляют выбором палитры, цветом обрамляющей экран рамки, преобразованием байта атрибута в данные о цвете символа и фона, а также режимом прокрутки экрана. Для указания адреса используется порт 3C0h. Для передачи данных также применяется порт 3C0h. Для определения текущего режима порта 3C0h (адрес индекса или передача данных) следует прочитать порт 3DAh. Кроме того, можно ориентироваться на цикл работы данного порта: первое обращение к порту указывает на номер индексного регистра, а второе — на передачу данных. При третьем обращении порт 3C0h опять ожидает номера индексного регистра. И так далее. Для считывания данных может применяться порт 3C1h. Формат регистра 3C0h показан в табл. 4.25.

Приведем краткое описание таблицы.

- Биты 0—4 определяют номер индексного регистра, для которого должна быть выполнена операция чтения или записи.

- Бит 5, установленный в 1, закрывает доступ к внутренней палитре. Если бит установлен в 0, будет выполнена загрузка значений цветов во внутренние регистры палитры.

Таблица 4.25. *Формат регистра 3C0h*

Бит	Описание
0—4	Номер индексного регистра
5	Использование палитры
6—7	Резерв

Контроллер атрибутов поддерживает 20 индексных регистров (от 00h до 14h). Каждый индексный регистр отвечает за определенные настройки. Список индексных регистров показан в табл. 4.26.

Таблица 4.26. *Список индексных регистров контроллера атрибутов*

Код индексного регистра	Описание
00h—0Fh	Регистры палитры
10h	Регистр управления режимом
11h	Регистр управления цветом рамки
12h	Регистр управления цветовыми плоскостями
13h	Регистр горизонтального панорамирования в единицах пиксела
14h	Регистр выбора цвета

Рассмотрим подробнее индексные регистры контроллера атрибутов.

4.3.3.1. Регистры палитры (*00h—0Fh*)

Размер каждого регистра равен 8 бит. Регистры палитры предназначены для динамического преобразования атрибута текстового символа или значения цвета (в графическом режиме) в реальный код цвета на экране. Используются только биты с 0 по 5. При установке бита в 1 будет выбран соответствующий цвет. Изменять значения внутренних регистров палитры можно только во время обратного хода луча по вертикали, иначе возникнут проблемы с выводимым на экран изображением.

4.3.3.2. Регистр *10h*

Размер регистра равен 8 бит. Он предназначен для установки различных режимов работы. Формат регистра показан в табл. 4.27.

Таблица 4.27. *Формат регистра управления режимом (10h)*

Бит	Описание
0	Управление графическим режимом
1	Эмуляция монохромного дисплея
2	Использование псевдографики
3	Управление миганием символа
4	Резерв
5	Режим поэлементного панорамирования
6	Управление цветом
7	Управление регистрами палитры

Приведем краткие пояснения к таблице.

- Бит 0, установленный в 1, позволяет выбрать графический режим работы.
- Бит 1, установленный в 1, позволяет выбрать монохромный режим работы дисплея. Если бит установлен в 0, будет использоваться цветной режим.
- Бит 2 применяется в 9-разрядных символьных режимах для того, чтобы правильно отобразить непрерывную строку символов. При установке бита в 1 каждый 9-й столбец будет закрашен фоновым цветом остальных символов.
- Бит 3, установленный в 1, разрешает мигание символа. Если этот бит равен 0, используется насыщенность цвета фона символа (7 бит байта атрибута).
- Бит 5, установленный в 1, разрешает использование режима поэлементного панорамирования.
- Бит 6, установленный в 1, позволяет выбрать 8-битный цвет (256 цветов). Во всех остальных режимах этот бит должен быть установлен в 0.
- Бит 7, установленный в 1, позволяет использовать биты 4 и 5 регистра палитры вместо битов 0 и 1 регистра выбора цвета (14h).

4.3.3.3. Регистр *11h*

Размер регистра равен 8 бит. Он предназначен для установки цвета обрамляющей экран рамки. Используются все биты. Для установки цвета следует записать в этот регистр номер одного из регистров ЦАП (00h—FFh).

4.3.3.4. Регистр *12h*

Размер регистра равен 8 бит. Он позволяет устанавливать доступ к определенным цветовым плоскостям. Используются биты с 0 по 3. Установка в 1 какого-либо из этих битов, позволит открыть доступ к плоскости памяти, имеющей соответствующий номер (от 0 до 3).

4.3.3.5. Регистр *13h*

Размер регистра равен 8 бит. Он позволяет управлять горизонтальным панорамированием (сдвигом по горизонтали вправо) и используется битовое поле с 0 по 3. Для указания нового значения следует записать в это поле целое число, измеряемое в пикселах. Сдвиг может применяться в текстовых и графических режимах. Запись в регистр нужно производить в момент обратного хода луча по вертикали.

4.3.3.6. Регистр *14h*

Размер регистра равен 8 бит. Он позволяет расширить 6-разрядные значения цвета до 8-разрядных, а также заместить биты 4 и 5 регистров палитры. Формат регистра показан в табл. 4.28.

Таблица 4.28. *Формат регистра управления режимом* (`14h`)

Бит	Описание
0—1	Замещение 4 и 5 битов регистров палитры
2—3	Добавление 6 и 7 битов описания цвета
4—7	Резерв

Приведем краткое описание таблицы.

- Биты 0 и 1 управляют замещением 4 и 5 битов регистра палитры. Если в регистре управления режимом (`10h`) бит 7 установлен в 1, то 4 и 5 биты во всех регистрах палитры будут замещены 0 и 1 битом регистра выбора цвета.
- Биты 2 и 3 позволяют добавить два старших бита (6 и 7) к 5-разрядному представлению цвета для согласования с регистром цвета ЦАП. Этот режим может использоваться для быстрого переключения цветов ЦАП.

Рассмотрим пример работы с регистрами контроллера атрибутов. Попробуем установить графический режим работы с помощью регистра управления режимом (`10h`), как показано в листинге 4.39.

Листинг 4.39. Установка графического режима работы

```
xor AL, AL   ; обнуляем регистр
mov AL, 10h  ; записываем значение регистра 10h
mov DX, 03C0h; выбираем порт для записи индекса
out DX, AL   ; записываем значение в порт
mov DX, 03C0h; выбираем порт для записи индекса
in  BL, DX   ; читаем из порта 3C0h
xor BL, BL   ; обнуляем регистр
and BL, 01b  ; устанавливаем графический режим
mov AL, 10h  ; записываем значение регистра 10h
mov DX, 03C0h; выбираем порт для записи индекса
out DX, AL   ; записываем значение в порт
mov DX, 03C0h; выбираем порт для записи индекса
out DX, AL   ; записываем значение в порт
mov AL, BL   ; копируем значение
out DX, AL   ; записываем значение в порт
```

Приведенный выше пример сознательно упрощен для лучшего понимания работы с контроллером атрибутов. Аналогичный пример для C++ показан в листинге 4.40.

Листинг 4.40. Установка графического режима работы в C++

```
// переменная для хранения результата
DWORD dwResult  = 0;
// записываем значение регистра 10h в порт 03C0h
outPort (0x03C0, 0x10, 1);
// читаем из порта 3C0h
inPort ( 0x03C0, &dwResult, 1);
// устанавливаем графический режим
dwResult &= 0x01;
// записываем значение регистра 10h в порт 03C0h
outPort (0x03C0, 0x10, 1);
// записываем новое значение
outPort (0x03C0, dwResult, 1);
```

4.3.4. Регистры контроллера CRT

Контроллер катодно-лучевой трубки управляет кадровой разверткой (формированием кадров на дисплее), а также параметрами горизонтальной развертки. Для доступа к контроллеру CRT используются два основных порта: выбора адреса (порт 3D4h) и ввода-вывода данных (порт 3D5h). Кроме этого, контроллер поддерживает 25 индексных регистров, управляющих различ-

ными функциями контроллера CRT. Полный список этих регистров приведен в табл. 4.29. Для записи или чтения данных в порт 3D4h следует записать номер индексного регистра, а затем передать или получить данные через порт 3D5h.

Таблица 4.29. *Список индексных регистров контроллера CRT*

Код индексного регистра	Описание
00h	Регистр полной длины горизонтальной развертки
01h	Регистр длины горизонтальной развертки
02h	Начало хода луча по горизонтали
03h	Конец хода луча по горизонтали
04h	Начало обратного хода луча по горизонтали
05h	Конец обратного хода луча по горизонтали
06h	Количество линий растра экрана или длина вертикального хода луча
07h	Регистр переполнения
08h	Регистр предварительной развертки по горизонтали
09h	Максимальная высота строки развертки
0Ah	Регистр начальной позиции курсора
0Bh	Регистр конечной позиции курсора
0Ch	Старший байт в начальном адресе
0Dh	Младший байт в начальном адресе
0Eh	Старший байт в позиции курсора
0Fh	Младший байт в позиции курсора
10h	Начало обратного хода луча по вертикали
11h	Конец обратного хода луча по вертикали
12h	Количество линий в растре
13h	Регистр смещения
14h	Позиция символа подчеркивания
15h	Начало хода луча по вертикали
16h	Конец хода луча по вертикали
17h	Регистр управления режимом
18h	Регистр сравнения строк

Работать с регистрами CRT следует очень осторожно, чтобы не повредить монитор. Для защиты регистров 00h—07h от записи в регистре 11h бит 7 всегда установлен в 1. Чтобы восстановить возможность записи, следует обнулить этот бит.

4.3.4.1. Регистр *00h*

Размер регистра равен 8 бит. Он предназначен для определения полного размера горизонтальной развертки. Позволяет управлять частотой обновления экрана по горизонтали, устанавливая необходимое для этого количество времени.

4.3.4.2. Регистр *01h*

Размер регистра равен 8 бит. Он предназначен для управления длиной горизонтальной развертки, отображаемой на экране. Данный регистр устанавливает порядок вывода на экран значений пикселов из памяти дисплея. Начало вывода устанавливается количеством кодовых комбинаций активного дисплея минус один.

4.3.4.3. Регистр *02h*

Размер регистра равен 8 бит. Он предназначен для управления началом момента гашения луча по горизонтали. Данный регистр используется совместно с регистром 03h для получения точки гашения луча по горизонтали.

4.3.4.4. Регистр *03h*

Размер регистра равен 8 бит. Он предназначен для управления окончательным моментом гашения луча по горизонтали. Формат регистра показан в табл. 4.30.

Таблица 4.30. *Формат регистра 03h*

Бит	Описание
0—4	Указывает на конец гашения луча
5—6	Определяет количество импульсов задержки (как правило, равно 0)
7	Используется для поддержки светового пера (оставлено ради совместимости)

4.3.4.5. Регистр *04h*

Размер регистра равен 8 бит. Он позволяет управлять началом обратного хода луча по горизонтали. Значение этого регистра определяет временной

интервал, через который начнется передача синхронизирующих импульсов дисплею. Регистр используется совместно с регистром 05h.

4.3.4.6. Регистр *05h*

Размер регистра равен 8 бит. Он предназначен для управления окончательным моментом гашения луча по горизонтали. Формат регистра показан в табл. 4.31.

Таблица 4.31. *Формат регистра* 05h

Бит	Описание
0—4	Указывает на конец обратного хода луча
5—6	Определяет время задержки (как правило, равно 0)
7	Содержит 5 бит дополнительно к битам 0—4

4.3.4.7. Регистр *06h*

Размер регистра 10 битов. Старшие два бита (8 и 9) располагаются в регистре переполнения (07h). Регистр определяет количество вертикальных линий растра для активного экрана, иначе говоря, управляет значением длины обратного хода луча по вертикали.

4.3.4.8. Регистр *07h*

Размер регистра равен 8 бит. Он предоставляет дополнительные биты для различных управляющих регистров. Формат данного регистра показан в табл. 4.32.

Таблица 4.32. *Формат регистра* 07h

Бит	Описание
0	Бит 8 для регистра 06h
1	Бит 8 для регистра 12h
2	Бит 8 для регистра 10h
3	Бит 8 для регистра 15h
4	Бит 8 для регистра 18h
5	Бит 9 для регистра 06h
6	Бит 9 для регистра 12h
7	Бит 9 для регистра 10h

4.3.4.9. Регистр *08h*

Размер регистра равен 8 бит. Он позволяет настроить горизонтальную развертку. Кроме того, данный регистр позволяет в текстовом режиме организовать плавную прокрутку экрана. Формат регистра представлен в табл. 4.33.

Таблица 4.33. *Формат регистра* 08h

Бит	Описание
0—4	Количество линий вертикальной развертки
5—6	Управление позицией левого верхнего угла в видеопамяти
7	Резерв

Приведем краткое пояснение к таблице.

- Биты 0—4 содержат количество линий горизонтальной развертки, на которое экран должен быть прокручен вверх. Возможные значения должны лежать в диапазоне от 0 до максимальной высоты строки развертки *(см. разд. 4.3.4.10)*.
- Биты 5—6 используются совместно с регистрами начального адреса для получения координат левого верхнего пиксела или символа на экране.

4.3.4.10. Регистр *09h*

Размер регистра равен 8 бит. Он предназначен для управления высотой строки развертки. Кроме этого, регистр содержит дополнительные биты для некоторых других регистров. Формат данного регистра показан в табл. 4.34.

Таблица 4.34. *Формат регистра* 09h

Бит	Описание
0—4	Максимальная высота строки развертки
5	Бит 9 для регистра 15h
6	Бит 9 для регистра 18h
7	Режим двойного сканирования

Приведем краткое описание таблицы.

- Биты 0—4 определяют максимальное значение высоты одной строки развертки, иначе говоря, высоту символов на экране дисплея.
- Бит 5 определяет дополнительный старший (бит 9) разряд для регистра 15h.

- Бит 6 определяет дополнительный старший (бит 9) разряд для регистра 18h.
- Бит 7 управляет режимом двойного сканирования. При установке этого бита в 1 будут использоваться 400 линий. Если бит установлен в 0, то 200 линий.

4.3.4.11. Регистр *0Ah*

Размер регистра равен 8 бит. Он предназначен для управления начальной позицией курсора на экране. Формат регистра показан в табл. 4.35. Используется для текстового режима.

Таблица 4.35. *Формат регистра* 0Ah

Бит	Описание
0—4	Начальная позиция линии курсора
5	Управление курсором
6—7	Резерв

Приведем краткий комментарий к таблице.

- Биты 0—4 содержат значение позиции для линии развертки, на которой будет расположен курсор.
- Бит 5 управляет отображением курсора в текстовом режиме. Установка этого бита в 1 скрывает курсор с экрана.

4.3.4.12. Регистр *0Bh*

Размер регистра равен 8 бит. Он предназначен для управления конечной позицией курсора на экране. Формат регистра показан в табл. 4.36. Регистр используется для текстового режима.

Таблица 4.36. *Формат регистра* 0Bh

Бит	Описание
0—4	Конечная позиция линии курсора
5—6	Смещение курсора
7	Резерв

Приведем краткое описание таблицы.

- Биты 0—4 содержат значение позиции для линии развертки, на которой будет расположен курсор.

- Биты 5—6 используются в режиме EGA для синхронизации курсора с работой дисплея. В режиме VGA могут быть установлены в ноль или дополнять биты 0—4.

4.3.4.13. Регистр *0Ch*

Размер регистра равен 8 бит. Он содержит значение старшего байта (биты 8—15) начального адреса в видеопамяти, используемого для вывода информации на экран. Проще говоря, данный регистр содержит старшую часть адреса, которая определяет начальную позицию (левый верхний угол) на экране.

4.3.4.14. Регистр *0Dh*

Размер регистра равен 8 бит. Он содержит значение младшего байта (биты 0—7) начального адреса в видеопамяти, используемого для вывода информации на экран. Совместно с регистром 0Ch он определяет полный 16-битный адрес начальной позиции (левый верхний угол) на экране. Поскольку размер адресуемой памяти ограничен (только 16 разрядов), данные регистры могут применяться только для текстовых и для некоторых графических режимов.

4.3.4.15. Регистр *0Eh*

Размер регистра равен 8 бит. Он содержит значение старшего байта (биты 8—15) позиции курсора на экране.

4.3.4.16. Регистр *0Fh*

Размер регистра равен 8 бит. Он содержит значение младшего байта (биты 0—7) позиции курсора на экране. Совместно с регистром 0Eh он определяет позицию курсора на экране относительно левого верхнего угла. Оба регистра используются в текстовых режимах.

4.3.4.17. Регистр *10h*

Размер регистра равен 10 битов. Два старших бита (8 и 9) расположены в регистре переполнения (07h). Он предназначен для управления начальным значением обратного хода луча по вертикали.

4.3.4.18. Регистр *11h*

Размер регистра равен 8 бит. Он предназначен для управления конечным значением обратного хода луча по вертикали. Кроме этого, регистр позволяет блокировать запись в регистры 00h—07h. Формат регистра показан в табл. 4.37.

Таблица 4.37. *Формат регистра* `11h`

Бит	Описание
0—3	Обратный ход луча по вертикали
4—5	Резерв
6	Обновление памяти
7	Блокировка регистров `00h—07h`

Приведем краткие пояснения к таблице.

- Биты 0—3 определяют значение обратного хода луча по вертикали.
- Бит 6 управляет количеством циклов обновлений (регенерации) динамической памяти за время вертикального хода луча. Возможны 2 значения: 3 цикла (31,5 кГц), если бит равен 0, и 5 циклов (15,7 кГц), если бит равен 1.
- Бит 7 используется для защиты регистров `00h—07h`. Если бит установлен в 1, запись вышеуказанных регистров будет невозможна, за исключением бита 4 регистра переполнения (`07h`).

4.3.4.19. Регистр *12h*

Размер регистра равен 10 бит. Два старших бита (8 и 9) расположены в регистре переполнения (`07h`). Регистр содержит количество вертикальных линий в растре минус один для активного экрана.

4.3.4.20. Регистр *13h*

Размер регистра равен 8 бит. Он определяет смещение для ширины логического экрана в памяти. Для текстовых режимов значение данного регистра равно ширине экрана (в пикселах), деленной на двойной размер памяти. Для графических режимов смещение равно ширине экрана (в пикселах), деленной на произведение двойного размера памяти и количества пикселов в одном адресе памяти. Данный регистр позволяет создать виртуальный экран, превышающий физический размер.

4.3.4.21. Регистр *14h*

Размер регистра равен 8 бит. Он предназначен для управления символом подчеркивания. Формат регистра показан в табл. 4.38.

Приведем краткое описание таблицы.

- Биты 0—4 определяют позицию на горизонтальной строке развертки минус 1.

- Бит 5, установленный в 1, позволяет синхронизировать значение счетчика импульсов со значением синхроимпульсов, деленным на 4.
- Бит 6, установленный в 1, позволит использовать для адресации памяти двойные слова (32 бита).

Таблица 4.38. *Формат регистра* `14h`

Бит	Описание
0—4	Позиция подчеркивания
5	Счетчик
6	Множитель
7	Резерв

4.3.4.22. Регистр *15h*

Размер регистра равен 10 бит. Бит 8 расположен в регистре `07h`, а бит 9 — в регистре `09h` (бит 5). Этот регистр позволяет управлять значением начального хода луча по вертикали.

4.3.4.23. Регистр *16h*

Размер регистра равен 8 бит, но используются только биты с 0 по 6. Он позволяет управлять значением конечного хода луча по вертикали.

4.3.4.24. Регистр *17h*

Размер регистра равен 8 бит. Он предназначен для управления различными режимами работы. Формат регистра показан в табл. 4.39.

Таблица 4.39. *Формат регистра* `17h`

Бит	Описание
0	Управление мультиплексором
1	Управление мультиплексором
2	Вертикальная синхронизация
3	Использование счетчика адреса
4	Резерв
5	Выбор адресов
6	Режим адресации
7	Управление синхронизацией

Приведем краткие пояснения к таблице.

- Бит 0 управляет битом 13 мультиплексора контроллера CRT. Если бит установлен в 1, будет выбран счетчик адресов. Бит применяется для поддержки более старых контроллеров.
- Бит 1 управляет битом 14 мультиплексора контроллера CRT. Если бит установлен в 1, будет выбран счетчик развертки по горизонтали.
- Бит 2 управляет вертикальной синхронизацией. Если бит установлен в 1, горизонтальные синхроимпульсы делятся на два, что позволяет удвоить разрешение по вертикали.
- Бит 3, установленный в 1, приводит к тому, что счетчик адресов делит на два входные кодовые импульсы символов.
- Бит 5 позволяет управлять битами адресов 13 и 15. Он используется для совместимости с некоторыми типами мониторов.
- Бит 6 позволяет установить желаемый режим адресации: байтами или словами. При установке этого бита в 0 будет использоваться адресация словами.
- Бит 7, установленный в 0, позволяет блокировать горизонтальные и вертикальные сигналы для обратного хода луча.

4.3.4.25. Регистр *18h*

Размер регистра равен 10 бит. Дополнительные биты 8 и 9 расположены в регистрах `07h` и `09h`, соответственно. Регистр определяет разделяющую строку развертки по горизонтали.

* * *

На этом можно завершить описание регистров CRT. Рассмотрим простой пример доступа к регистру `0Ah` для удаления курсора с экрана (листинг 4.41).

Листинг 4.41. Удаление курсора с экрана

```
xor AL, AL    ; обнуляем регистр
mov AL, 0Ah   ; записываем значение регистра 0Ah
mov DX, 03D4h; выбираем порт для записи индекса
out DX, AL    ; записываем значение в порт
xor BL, BL    ; обнуляем регистр
mov DX, 03D5h; выбираем порт для записи индекса
in  BL, DX    ; читаем из порта 3D5h
test BL, 20h ; если курсор уже скрыт
je  END_PROC ; выходим из процедуры
and BL, 20h   ; отключаем курсор
```

```
mov AL, 0Ah  ; записываем значение регистра 0Ah
mov DX, 03D4h; выбираем порт для записи индекса
out DX, AL   ; записываем значение в порт
mov DX, 03D5h; выбираем порт для записи индекса
mov AL, BL   ; копируем байт
out DX, AL   ; записываем значение в порт
```

Аналогичный пример для C++ представлен в листинге 4.42.

Листинг 4.42. Удаление курсора с экрана в C++

```
// переменная для хранения результата
DWORD dwResult  = 0;
// записываем значение регистра 0Ah в порт 03D4h
outPort (0x03D4, 0x0A, 1);
// читаем из порта 3D5h
inPort ( 0x03D5, &dwResult, 1);
// если курсор уже скрыт, выходим из процедуры
if ( ( dwResult & 0x20) == 1)
   return;
// отключаем курсор
dwResult &= 0x20;
// записываем значение регистра 0Ah в порт 03D4h
outPort (0x03D4, 0x0A, 1);
// записываем новое значение
outPort (0x03D5, dwResult, 1);
```

Большинство из описываемых регистров CRT используются попарно (например, регистры 0Ch и 0Dh или 0Eh и 0Fh).

4.3.5. Регистры ЦАП

Регистры ЦАП предназначены для преобразования цифровых данных, поступающих с видеоконтроллера, в аналоговый сигнал, который поступает на экран дисплея. ЦАП поддерживает 256 регистров (256 цветовых оттенков одновременно), каждый из которых содержит по три значения уровней для трех основных цветов: красного, зеленого и синего. Размер каждого регистра равен 18 бит (по 6 битов на каждый цвет). Значение каждого цвета может лежать в диапазоне от 0 до 63 единиц. Для доступа к регистрам ЦАП используются три порта:

1. Порт 3C7h в режиме записи управляет адресным регистром, а в режиме чтения считывает текущее состояние ЦАП.

2. Порт 3C8h позволяет в режиме записи указать номер регистра таблицы цветов, а в режиме чтения возвращает текущий номер регистра таблицы цветов.

3. Порт 3C9h в режиме записи позволяет установить новое значение цвета для выбранного регистра таблицы цветов, а в режиме чтения считывает текущее значение цвета из указанного регистра цветов.

Для записи нового значения цвета требуется выполнить подряд три операции, по одной на каждую составляющую цвета (красный, зеленый, синий). После выполнения третьей операции индекс текущего регистра будет автоматически увеличен на единицу, что упрощает запись всей таблицы цветов для 256 регистров. Рекомендуется перед записью или чтением регистров ЦАП отключать прерывания.

Для записи новых значений цветов вначале следует записать в порт 3C8h номер регистра таблицы цветов (от 0 до 256). После этого в регистр данных (порт 3C9h) записываются последовательно три значения оттенка цвета, по одному на каждый основной цвет (красный, зеленый, синий). Регистр адреса (3C8h) увеличивает номер индекса на 1, и можно сразу продолжить запись трех составляющих цвета в следующий регистр таблицы цветов. Учитывая эту особенность, имеет смысл за одну операцию записи установить новые значения цветов для всей таблицы (256 регистров). Запись значений палитры цветов необходимо выполнять во время обратного хода луча. Рассмотрим основные регистры ЦАП подробнее.

4.3.5.1. Регистр *3C7h*

Размер регистра равен 8 бит. В режиме чтения используются только 0 и 1 биты. Формат регистра в режиме записи показан в табл. 4.40, а в режиме чтения — в табл. 4.41.

Таблица 4.40. *Формат регистра* 3C7h *(запись)*

Бит	Описание
0—7	Номер регистра таблицы цветов (0—256)

Таблица 4.41. *Формат регистра* 3C7h *(чтение)*

Бит	Описание
0—1	Состояние
2—7	Резерв

В режиме чтения биты 0—1 определяют текущее состояние ЦАП: 0 — ЦАП находится в режиме чтения, 3 — ЦАП находится в режиме записи.

4.3.5.2. Регистр *3C8h*

Размер регистра равен 8 бит. Он используется для записи номера регистра палитры, для которого будут устанавливаться новые значения цветовых оттенков. В режиме чтения может вернуть текущий номер регистра палитры.

4.3.5.3. Регистр *3C9h*

Размер регистра равен 8 бит, но используются только младшие 6 разрядов. Он применяется для записи или чтения значения цвета из выбранного регистра палитры. Операция чтения или записи выполняется три раза: для красного, зеленого и синего цветов. Формат регистра в режиме записи показан в табл. 4.42.

Таблица 4.42. *Формат регистра* 3C9h *(чтение и запись)*

Бит	Описание
0—5	Значение цвета
6—7	Резерв

Существует еще один регистр, адресуемый через порт 3C6h. Он используется для маскирования значения цвета в одном из регистров таблицы цветов и доступен для записи и считывания. По умолчанию значение в порту равно FFh. При обращении к регистру будет выполнена операция AND между содержимым этого регистра и значением выбранного регистра таблицы цветов. Не рекомендуется писать в порт 3C6h, поскольку это может привести к разрушению таблицы цветов.

4.3.6. Регистры синхронизатора

Регистры синхронизатора управляют подстройкой данных, передаваемых ЦАП, а также позволяют корректировать работу цепей синхронизатора для программирования нестандартных видеорежимов. Вся работа с цепями синхронизатора построена на двух регистрах: на адресном (порт 3C4h) и на регистре данных (порт 3C5h). Они позволяют получить доступ к пяти (00h—04h) индексным регистрам. Перед началом работы следует записать в адресный регистр (3C4h) номер индексного регистра, а затем можно считывать или записывать данные через порт 3C5h. Рассмотрим индексные регистры подробнее.

4.3.6.1. Регистр *00h*

Размер регистра равен 8 бит, но используются только младшие два бита (0 и 1). Регистр управляет режимом сброса цепей синхронизации. Формат регистра показан в табл. 4.43.

Таблица 4.43. *Формат регистра* 00h

Бит	Описание
0	Асинхронный сброс
1	Синхронный сброс
2—7	Резерв

Приведем краткое описание таблицы.

- Бит 0, установленный в 0, позволяет выполнить асинхронный сброс цепей синхронизатора и остановить отправление синхроимпульсов. Для восстановления работы необходимо установить бит в 1.
- Бит 1, установленный в 0, позволяет выполнить синхронный сброс цепей синхронизатора и остановить отправление синхроимпульсов. Для восстановления работы необходимо установить бит в 1.

4.3.6.2. Регистр *01h*

Размер регистра равен 8 бит. Он предназначен для управления режимом синхронизации. Формат регистра представлен в табл. 4.44.

Таблица 4.44. *Формат регистра* 01h

Бит	Описание
0	Размер символов
1	Резерв
2	Частота преобразования
3	Частота обновления
4	Управление частотой преобразования
5	Блокировка дисплея
6—7	Резерв

Приведем краткое пояснение к таблице.

- Бит 0 позволяет установить размер символов. Возможны два значения: 8 точек (бит равен 1) и 9 точек (бит равен 0). Изменение размера символа может понадобиться при переключении текстовых режимов работы.
- Бит 2, установленный вместе с битом 4 в 0, определяет, что преобразование данных в выходных цепях происходит при каждом синхроимпульсе, иначе — через один.

- Бит 3, установленный в 0, определяет, что частота обновления точки (или символа) совпадает с частотой тактового генератора. Если бит равен 1, частота обновления будет уменьшена в 2 раза. Изменение частоты повлияет на все синхронизирующие импульсы, используемые в устройстве.
- Бит 4, установленный в 1, уменьшает частоту преобразования данных в 4 раза.
- Бит 5, установленный в 1, позволит выключить дисплей (блокировать передаваемые из видеопамяти данные). Может использоваться для полноэкранных модификаций изображения.

4.3.6.3. Регистр *02h*

Размер регистра равен 8 бит. Используются младшие четыре бита (0—3). Регистр позволяет блокировать до четырех плоскостей видеопамяти (0—3). Установка бита в 1 разрешает запись в соответствующую плоскость.

4.3.6.4. Регистр *03h*

Размер регистра равен 8 бит. Он предназначен для выбора шрифта. Формат регистра приведен в табл. 4.45.

Таблица 4.45. *Формат регистра* 03h

Бит	Описание
0—1	Шрифт В
2—3	Шрифт А
4	Выбор символа из шрифта В
5	Выбор символа из шрифта А
6—7	Резерв

Приведем краткое пояснение к таблице.

- Биты 0—1 позволяют выбрать шрифт В, если бит 3 атрибута символа установлен в 0.
- Биты 2—3 позволяют выбрать шрифт А, если бит 3 атрибута символа установлен в 1. Адрес шрифта в памяти имеет следующие значения: `000b` — `0000h—1FFFh`, `001b` — `4000h—5FFFh`, `010b` — `8000h—9FFFh`, `011b` — `C000h—DFFFh`, `100b` — `2000h—3FFFh`, `101b` — `6000h—7FFFh`, `110b` — `A000h—BFFFh`.
- Бит 4 определяет бит 2 для поля шрифта В.
- Бит 5 определяет бит 2 для поля шрифта А.

4.3.6.5. Регистр *04h*

Размер регистра равен 8 бит. Он позволяет управлять режимом работы видеопамяти. Формат регистра показан в табл. 4.46.

Таблица 4.46. *Формат регистра* 04h

Бит	Описание
0	Резерв
1	Расширенная память
2	Выбор адресов
3	Управление операциями чтения
4—7	Резерв

Приведем краткое описание таблицы.

- Бит 1, установленный в 1, позволяет расширить объем адресуемой памяти от 64 до 256 Кбайт.
- Бит 2 управляет доступом к четным и нечетным плоскостям в памяти. При установке бита в 0 четные адреса работают с плоскостями 0 и 2, а нечетные — с 1 и 3.
- Бит 3, установленный в 0, позволяет последовательно считывать адресуемые данные из памяти.

Рассмотрим упрощенный пример работы с регистрами синхронизатора для выполнения синхронного сброса (листинг 4.43).

Листинг 4.43. Выполнение синхронного сброса

```
xor AL, AL   ; обнуляем регистр
mov AL, 00h  ; записываем значение регистра 00h
mov DX, 03C4h; выбираем порт для записи индекса
out DX, AL   ; записываем значение в порт
mov DX, 03C5h; выбираем порт для записи данных
mov AL, 1    ; синхронный сброс
out DX, AL   ; записываем значение в порт
; восстанавливаем синхронизацию
mov AL, 00h  ; записываем значение регистра 00h
mov DX, 03C4h; выбираем порт для записи индекса
out DX, AL   ; записываем значение в порт
mov DX, 03C5h; выбираем порт для записи данных
mov AL, 3    ; восстанавливаем сигнал синхронизации
out DX, AL   ; записываем значение в порт
```

Аналогичный пример для C++ показан в листинге 4.44.

Листинг 4.44. Выполнение синхронного сброса в C++

```
// переменная для хранения результата
DWORD dwResult  = 0;
// записываем значение регистра 00h в порт 03C4h
outPort (0x03C4, 0x00, 1);
// синхронный сброс
outPort (0x03C5, 0x01, 1);
// восстанавливаем синхронизацию
// записываем значение регистра 00h в порт 03C4h
outPort (0x03C4, 0x00, 1);
// восстанавливаем сигнал синхронизации
outPort (0x03C5, 0x03, 1);
```

На этом примере можно завершить программирование видеоконтроллера с использованием аппаратных портов и перейти к рассмотрению возможностей интерфейса Win32 API для работы с видеоадаптером и монитором.

4.4. Использование Win32 API

Программисту, работающему с интерфейсом Win32, практически не нужно думать о непосредственном взаимодействии с видеоконтроллером и монитором. Интерфейс предоставляет универсальные законченные решения для вывода графической и текстовой информации на экран. Этих возможностей вполне достаточно для разработки большинства пользовательских программ. Даже для написания игровых приложений существуют дополнительные библиотеки (например, DirectX), которые в тандеме с функциями Win32 API позволяют решить практически любые задачи. При этом скорость работы таких программ остается достаточно высокой, а если учесть мощь современных процессоров (в том числе и графических), то вполне понятно, почему разработчики предпочитают стандартные библиотеки Win32 API непосредственному программированию аппаратных портов.

Здесь мы не будем изучать программирование графики в Windows, а рассмотрим только возможности управления видеоконтроллером и монитором, любезно "сохраненные" фирмой Microsoft для разработчиков программного обеспечения:

1. Управление графическими режимами.
2. Получение информации о возможностях видеоконтроллера.
3. Управление монитором в Windows.

4.4.1. Управление графическими режимами

В Windows существует возможность программно установить любой поддерживаемый видеоадаптером режим работы. Для этого сначала нужно получить список всех поддерживаемых режимов, а затем активизировать его. Сразу хочу заметить, что видеоадаптер, как правило, поддерживает гораздо больше режимов, чем установленный монитор, поэтому важно перед установкой нового режима точно знать, будет ли с ним работать монитор. Игнорирование этого правила может привести даже к выходу из строя монитора. Программист обычно не может заранее знать аппаратную конфигурацию потенциального пользователя, поэтому вся ответственность выбора графического режима ложится на пользователя. Однако рекомендую программистам перед установкой нового режима выводить предупреждающие сообщения, активизировать новый режим на несколько секунд (10—20), после чего возвращаться к предыдущему. Это позволит уберечь начинающего пользователя от грубых ошибок и добавит вашей программе только плюсы.

Для получения поддерживаемых режимов используется функция `EnumDisplaySettings`. Она имеет три аргумента. Первый аргумент является указателем на строковое значение с именем устройства вывода. Для Windows 95/98/ME это значение должно быть установлено в `NULL`. Для Windows NT/2000/XP/SR3 можно указать параметр `DeviceName` структуры `DISPLAY_DEVICE`. Данная структура позволяет сохранять различные параметры для устройства вывода, используется в функции `EnumDisplayDevices` и будет рассмотрена далее в этой главе. Второй аргумент функции `EnumDisplaySettings` определяет тип возвращаемой информации и может принимать одно из следующих значений: `ENUM_CURRENT_SETTINGS` (получить текущие параметры) или `ENUM_REGISTRY_SETTINGS` (получить параметры из реестра). Кроме того, этот аргумент является своеобразным счетчиком. Данная функция должна вызываться определенное число раз (по одному для каждого поддерживаемого режима), и второй аргумент увеличивает значение счетчика при каждом вызове функции, начиная с 0. Последний аргумент функции является указателем на структуру `DEVMODE`, в которую записывается информация о найденном режиме. Формат данной структуры очень велик и многие ее поля не имеют прямого отношения к информации о режиме, поэтому приведу описание только нужных нам полей.

- Поле `dmBitsPerPel` предназначено для хранения информации о глубине цвета (в битах) для найденного режима: 4 бита для 16 цветов, 8 — для 256 цветов, 16 — для 65 536 и т. д.
- Поле `dmPelsWidth` определяет разрешение по горизонтали в пикселах.
- Поле `dmPelsHeight` определяет разрешение по вертикали в пикселах.
- Поле `dmDisplayFrequency` определяет частоту кадровой (вертикальной) развертки (частота обновления экрана) в герцах. Данное поле не поддерживается в Windows 95/98/ME.

Кроме того, перед использованием структуры DEVMODE следует записать в поле dmSize размер самой структуры (sizeof(DEVMODE)), а в поле dmDriverExtra — значение 0. Теперь у нас есть все необходимые составляющие, чтобы написать собственную функцию для получения всех поддерживаемых устройством режимов. Пример такой функции показан в листинге 4.45.

Листинг 4.45. Перечисление всех поддерживаемых видеорежимов

```
// определим глобальную переменную
DEVMODE* dvMode = NULL; // указатель на структуру DEVMODE
// назовем нашу функцию GetDisplayModes
void GetDisplayModes ( HWND hComboBox)
{
// счетчик для функции EnumDisplaySettings
DWORD dwNumberMode = ( DWORD) -1;
// буфер форматирования и вывода информации
TCHAR tszMode [100];
// выделяем необходимое количество памяти
dvMode = new DEVMODE [250]; // максимум 250 режимов, если найдутся
// если нет свободной памяти, завершаем функцию
if ( dvMode == NULL) return;
// пишем цикл для поиска всех возможных режимов
do {
     // увеличиваем счетчик на единицу
     dwNumberMode ++;
     // заполняем требуемые поля структуры DEVMODE
     dvMode [dwNumberMode].dmSize = sizeof ( DEVMODE);
     dvMode [dwNumberMode].dmDriverExtra = 0;
} while ( EnumDisplaySettings ( NULL, dwNumberMode,
                                  & dvMode [dwNumberMode]));
// форматируем полученные данные
 for ( int i = 0; i < dwNumberMode; i++)
 {
  // если в поле dmDisplayFrequency стоит 0 или 1, игнорируем его
  if (dvMode [i].dmDisplayFrequency == 0 ||
      dvMode [i].dmDisplayFrequency == 1) // Windows 95-98-ME
    {
      sprintf ( tszMode, __TEXT ( "%d x %d (%d bit),  0 Hz"),
                dvMode [i].dmPelsWidth, dvMode [i].dmPelsHeight,
                dvMode [i].dmBitsPerPel);
    }
  else // Windows NT-2000-XP-2003
```

```
    {
       wsprintf ( tszMode, __TEXT ("%d x %d (%d bit), %d Hz"),
                  dvMode [i].dmPelsWidth, dvMode [i].dmPelsHeight,
                  dvMode [i].dmBitsPerPel, dvMode [i].dmDisplayFrequency);
    } // end if
  // записываем отформатированную строку во внешний список
  SendMessage ( hComboBox, CB_ADDSTRING, 0, (LPARAM) (LPCTSTR) tszMode);
 } // end for
// выбираем в списке самый верхний пункт
SendMessage ( hComboBox, CB_SETCURSEL, 0, 0);
// освобождаем память
if ( dvMode) delete [] dvMode;
} // выходим из функции
```

Как видно из этого примера, наша функция `GetDisplayModes` имеет всего один аргумент — дескриптор раскрывающегося списка (`ComboBox`). После выполнения функции в списке будут размещены все найденные режимы. Поскольку некоторые системы (Windows 95, 98 и ME) не возвращают значение вертикальной частоты, мы использовали дополнительное условие, в котором проверяли значение поля `dmDisplayFrequency`. Если оно равно 0 или 1, значит, операционная система не поддерживает получение частоты. Для надежности можно проверить текущую версию Windows с помощью функции `GetVersion`. Рекомендую всем читателям в профессиональных приложениях использовать именно такой вариант.

Созданная нами функция прекрасно справляется с поставленной задачей, но имеет один неприятный сюрприз: информация о поддерживаемых режимах выводится не по порядку, а хаотично. Этот минус легко исправить, добавив код сортировки найденных режимов (например, с помощью функции `qsort`).

После того как мы определили все режимы, можно использовать функцию `ChangeDisplaySettings` для установки нового режима. Функция имеет всего два аргумента, первый из которых является указателем на структуру `DEVMODE`, а второй определяет флаг выполнения операции. Список всех флагов приведен в табл. 4.47.

Таблица 4.47. *Список флагов функции* `ChangeDisplaySettings`

Флаг	**Описание**
`0`	Смена режима будет выполнена динамически
`CDS_TEST`	Пробная установка требуемого режима
`CDS_UPDATEREGISTRY`	Смена режима будет выполнена динамически и сохранена в системном реестре

Таблица 4.47 (окончание)

Флаг	Описание
CDS_SET_PRIMARY	Позволяет выбрать текущее устройство вывода на экран первичным
CDS_RESET	Смена режима будет выполнена, даже если требуемый режим уже установлен ранее

После выполнения функция ChangeDisplaySettings возвращает определенное значение, по которому легко узнать результат операции. При успешной смене режима будет возвращено значение DISP_CHANGE_SUCCESSFUL. Если функция возвратит значение DISP_CHANGE_RESTART, то необходима перезагрузка компьютера для активизации выбранного графического режима. В случае невозможности установить требуемый режим функция вернет значение DISP_CHANGE_FAILED, а если указанный графический режим не поддерживается, то DISP_CHANGE_BADMODE.

А теперь рассмотрим на практике установку нового графического режима в Windows. Для этого напишем код, показанный в листинге 4.46.

Листинг 4.46. Установка нового графического режима

```
// назовем функцию SetDisplayModes
int SetDisplayModes ( DEVMODE* mode)
(
LONG lResult = 0;
// проверяем указатель
if ( mode == NULL) return -1;
// устанавливаем новый режим дисплея
lResult = ChangeDisplaySettings ( mode, CDS_UPDATEREGISTRY);
// обрабатываем результат операции
switch ( lResult)
{
    case DISP_CHANGE_SUCCESSFUL:
      // операция успешно завершена
      return 0;
    case DISP_CHANGE_FAILED:
      // не удалось установить требуемый режим
      return 1;
    case DISP_CHANGE_RESTART:
      // требуется выполнить перезагрузку ПК для активации режима
      return 2;
}
```

```
// неизвестная ошибка
return -1;
}
// теперь можно вызвать нашу функцию для установки режима
SetDisplayModes ( &dvMode [2]); // выбираем из списка режим 2
```

Поскольку указатель на структуру DEVMODE мы сделали глобальным, можно вызывать нашу функцию SetDisplayModes в любом удобном месте. В качестве аргумента ей передается указатель на DEVMODE. Номер выбранного нами режима (2) абсолютно случаен. Кроме того, еще раз замечу, что описания режимов в массиве структур DEVMODE расположены хаотично, а потому нужно быть очень внимательным при передаче номера режима, а лучше отсортировать все режимы по порядку.

4.4.2. Проверка возможностей видеоадаптера

В Windows существует удобная возможность получить различную полезную информацию о параметрах текущего графического режима, а также совместимости установленного видеоадаптера с различными графическими функциями рисования и управления цветом. Со всеми этими задачами справляется одна-единственная функция GetDeviceCaps. Она имеет всего два аргумента: первый определяет контекст устройства (для получения текущих параметров можно установить в NULL), а второй аргумент определяет флаг для требуемого параметра. Список интересующих нас флагов представлен в табл. 4.48. Возвращаемое значение функции зависит от установленного флага.

Таблица 4.48. *Список флагов функции* GetDeviceCaps

Флаг	Описание
DRIVERVERSION	Позволяет получить версию драйвера устройства
ASPECTX	Относительная ширина пиксела устройства для рисования линии
ASPECTY	Относительная высота пиксела устройства для рисования линии
ASPECTXY	Ширина пиксела устройства по диагонали для рисования линии
BITSPIXEL	Число граничных битов для отображения цвета каждого пиксела
CLIPCAPS	Поддержка устройством прямоугольных областей отсечения (1 — есть, 0 — нет)
COLORRES	Фактическое разрешение по цвету (бит/пиксел). Следует применять только, если установлен бит RC_PALETTE для флага RASTERCAPS

Таблица 4.48 (окончание)

Флаг	Описание
CURVECAPS	Определяет поддержку вывода кривых линий (CC_NONE — нет, CC_CIRCLES — окружность, CC_ELLIPSES — эллипс, CC_ROUNDRECT — скругленный прямоугольник, CC_PIE — сектор, CC_CHORD — хорда). Возвращаемое значение является комбинацией возможных значений
HORZRES	Ширина экрана в пикселах
VERTRES	Высота экрана в строках растра
HORZSIZE	Ширина физического экрана в миллиметрах
VERTSIZE	Высота физического экрана в миллиметрах
LINECAPS	Определяет поддержку вывода кривых линий (LC_NONE — нет, LC_MARKER — маркер, LC_POLYLINE — ломаная линия, LC_WIDE — широкая линия). Возвращаемое значение является комбинацией возможных значений
LOGPIXELSX	Количество пикселов на одном логическом дюйме по ширине
LOGPIXELSY	Количество пикселов на одном логическом дюйме по высоте
NUMBRUSHES	Поддерживаемое число кистей
NUMPENS	Поддерживаемое число перьев
NUMFONTS	Поддерживаемое число шрифтов
PLANES	Количество цветовых плоскостей
NUMCOLORS	Если устройство не поддерживает цвет больше 8 битов на пиксел, будет возвращено число входов в таблице цветов, иначе –1
RASTERCAPS	Определяет поддержку растровых операций (RC_PALETTE — зависит от палитры, RC_BITBLT — растровые изображения, RC_SCALING — масштабирование). Возвращаемое значение является комбинацией возможных значений
SIZEPALETTE	Количество входов в системной палитры (должен быть установлен бит RC_PALETTE для флага RASTERCAPS)
VREFRESH	Частота вертикальной развертки в герцах (в Windows 95/98/ME не поддерживается)

Попробуем на примере воспользоваться данной функцией. В листинге 4.47 приводится удобный способ получения параметров текущего графического режима.

Листинг 4.47. Получение параметров для текущего графического режима

```
// объявляем переменные
HDC hDC = NULL; // дескриптор контекста устройства
```

```
// переменные для хранения параметров режима
DWORD dwX = 0, dwY = 0, dwBit = 0, dwHz = 0;
DWORD dwRaster = 0, dwColors = 0;
// получаем текущий контекст устройства
hDC = GetDC ( NULL);
dwX = GetDeviceCaps ( hDC, HORZRES); // разрешение по горизонтали
dwY = GetDeviceCaps ( hDC, VERTRES); // разрешение по вертикали
dwBit = GetDeviceCaps ( hDC, BITSPIXEL); // разрядность цвета
dwHz = GetDeviceCaps ( hDC, VREFRESH); // частота по вертикали
// проверяем поддержку растра
dwRaster = GetDeviceCaps ( hDC, RASTERCAPS);
// проверяем поддержку палитры
dwRaster = ( dwRaster & RC_PALETTE) ? true : false;
 if ( dwRaster) // палитра поддерживается
    // получаем число цветов
    dwColors = GetDeviceCaps ( hDC, SIZEPALETTE);
 else // палитра не поддерживается
    dwColors = GetDeviceCaps ( hDC, NUMCOLORS);
// освобождаем контекст устройства
ReleaseDC ( NULL, hDC);
```

4.4.3. Управление монитором

И последнее, с чем хочется познакомить читателя, — это возможности управления монитором. Выбор графических режимов мы уже разобрали, а теперь поговорим о поддержке Win32 API энергосберегающих возможностей монитора. Существует, как минимум, две операции по управлению питанием монитора: первая позволяет перевести монитор в ждущий режим, а вторая полностью выключает основные цепи питания, переводя устройство в режим сна. Во втором режиме потребление питания наименьшее. Сразу замечу, что описываемые возможности будут работать только с мониторами, поддерживающими энергосберегающие режимы (все современные устройства).

Для управления режимами мы будем применять функцию PostMessage. Для тех, кто не знаком с ней, скажу, что данная функция позволяет поместить любое сообщение в общую очередь, связанную с текущим процессом и возвратить управление без ожидания результата. Для управления монитором мы должны будем послать ему системное сообщение WM_SYSCOMMAND с дополнительным параметром SC_MONITORPOWER. Как это делается, можно посмотреть в листинге 4.48.

Листинг 4.48. Управление питанием монитора

```
// функция для перевода монитора в ждущий режим
void MonitorLowPower ()
```

```
{
   PostMessage ( HWND_BROADCAST, WM_SYSCOMMAND, SC_MONITORPOWER, 1L);
}
// функция для выключения монитора
void MonitorOffPower ()
{
   PostMessage ( HWND_BROADCAST, WM_SYSCOMMAND, SC_MONITORPOWER, 2L);
}
```

Как видите, все достаточно просто и эффективно. Первый аргумент функции HWND_BROADCAST указывает на то, что сообщение должно быть послано всем окнам верхнего уровня (в том числе скрытым и зависшим) в системе. Второй аргумент определяет сообщение WM_SYSCOMMAND. Третий и четвертый аргументы описывают дополнительные параметры управления. В данном случае это специальное значение для управления питанием монитора и код операции (1 или 2).

На этом, думаю, можно закончить рассмотрение вопросов программирования видеоконтроллера и монитора.

ГЛАВА 5

Работа с видео

Пожалуй, возможность просмотра видеофильмов является второй после игрушек причиной покупки современного домашнего компьютера. И это не удивительно. С покупкой мультимедийного компьютера на второй план уходит видеомагнитофон с его громоздкими кассетами и посредственным качеством изображения и звука.

Для реализации поддержки видео в операционных системах Windows фирма Microsoft разработала собственный формат, ставший стандартом де-факто. Называется он AVI (Audio-Video Interleaved — формат с чередованием аудио- и видеоданных). Данный формат поддерживает видеоизображение и звук, синхронизированные между собой. Файл в формате AVI имеет расширение с таким же названием (avi) и содержит последовательность растровых изображений, а также один или более каналов звука. Формат AVI может использовать различные методы сжатия повторяющихся данных, что позволяет немного уменьшить конечный размер файла, но, тем не менее, обеспечивает отличное качество изображения и звука. Однако, несмотря на это, использование данного формата ограничено в основном предварительной обработкой (например, оцифровкой данных с видеокамеры с последующим редактированием) исходных видеоданных. Связано это с тем, что, даже несмотря на сжатие данных, выходной файл может иметь огромные размеры (в зависимости от разрешения и глубины цвета). Для записи же видеоданных на диски используются, как правило, другие форматы, обеспечивающие более высокую степень компрессии (например, MPEG 2, 4).

В любом случае, если вы работаете в Windows, так или иначе придется столкнуться с форматом AVI (оцифровка аналоговых данных, организация интерактивного интерфейса и т. д.). Поскольку этот формат является стандартным для Windows, фирма Microsoft позаботилась о его программной поддержке, добавив соответствующие возможности в Win32 API.

В этой главе мы попытаемся научиться работать с AVI-файлами посредством указанного выше интерфейса. Для этого мы воспользуемся двумя различными, как по сложности, так и по возможностям, способами:

1. Использование универсального интерфейса управления мультимедийными данными (MCI — Media Control Interface).
2. Использование набора функций поддержки видео в Windows (VFW – Video for Windows).

5.1. Использование MCI

Первый способ наиболее простой и позволяет быстро добавить в программу поддержку файлов в формате AVI, но имеет ограниченные возможности. Он идеально подходит для создания интерактивного интерфейса, а также различных анимационных эффектов. Кроме видеофайлов, он поддерживает аудиодиски (CDDA), файлы WAV и MIDI. Здесь мы рассмотрим только ту часть, которая относится к работе с видео, а в *гл. 7* познакомимся с программированием звука.

Для активизации интерфейса MCI необходимо добавить в проект заголовочные файлы MMsystem.h и Vfw.h, а также библиотеки Winmm.lib и Vfw.lib. Это необходимо для правильной инициализации функций, структур и констант, поддерживающих мультимедийные возможности Windows.

Чтобы воспроизвести AVI-файл, можно воспользоваться следующими макрокомандами:

- `MCIWndCreate`. Является функцией. Позволяет открыть устройство MCI или AVI-файл и зарегистрировать собственное окно для просмотра видеоданных;
- `MCIWndPlay`. Начинает воспроизведение текущего файла;
- `MCIWndHome`. Позволяет установить исходное состояние воспроизведения (устанавливает курсор чтения в начало файла);
- `MCIWndPause`. Приостанавливает текущее воспроизведение;
- `MCIWndResume`. Продолжает воспроизведение, остановленное командой `MCIWndPause`;
- `MCIWndStop`. Останавливает текущее воспроизведение;
- `MCIWndDestroy`. Необходимо вызвать эту команду для закрытия устройства MCI и закрытия окна просмотра;
- `MCIWndClose`. Данная макрокоманда также закрывает устройство MCI, но оставляет окно просмотра активным. В дальнейшем его можно использовать для повторного просмотра AVI-файла.

Существует еще ряд дополнительных макрокоманд, но в данном случае они нам не понадобятся. Прежде чем перейти к практике, разберем подробнее основную функцию поддержки видеоданных.

Функция `MCIWndCreate` содержит четыре аргумента, каждый из которых имеет определяющее значение: первый аргумент `hwndParent` указывает на дескриптор родительского окна (`HWND`). Второй аргумент `hInstance` содержит дескриптор текущего приложения (`HINSTANCE`). Третий аргумент `dwStyle` определяет комбинацию стандартных стилей для окна:

- `WS_VSCROLL` — добавить вертикальную линейку прокрутки;
- `WS_HSCROLL` — добавить горизонтальную линейку прокрутки;
- `WS_DISABLED` — отключить сообщения от мыши и клавиатуры;
- `WS_EX_CONTEXTHELP` — добавить контекстную справку;
- `WS_EX_MDICHILD` — создать дочернее окно;
- `WS_EX_TOPMOST` — расположить поверх всех видимых окон.

Дополнительно к ним можно указать специфические стили, используемые только в этой функции:

- `MCIWNDF_NOAUTOSIZEWINDOW` — не изменять размеры окна при изменении размеров отображаемых видеоданных;
- `MCIWNDF_NOERRORDLG` — отключить сообщения об ошибках MCI;
- `MCIWNDF_NOOPEN` — блокировать команды меню для открытия файла;
- `MCIWNDF_NOPLAYBAR` — скрывать панель инструментов;
- `MCIWNDF_NOTIFYPOS` — уведомлять о текущей позиции воспроизведения;
- `MCIWNDF_NOTIFYALL` — использовать все доступные уведомления;
- `MCIWNDF_SHOWPOS` — отображать текущую позицию в области заголовка.

Последний четвертый аргумент функции `szFile` содержит указатель на текстовую строку с нулевым символом на конце (`LPSTR`), которая хранит имя устройства MCI или AVI-файла. В случае успешного завершения функция `MCIWndCreate` возвращает дескриптор окна (`HWND`) для вывода видеоданных. Полученный дескриптор окна понадобится нам в качестве основного аргумента для используемых в дальнейшем макрокоманд.

Рассмотрим пример программы, которая позволяет воспроизводить видеофайлы в формате AVI или MPG в главном окне. Основной код программы представлен в листинге 5.1.

Листинг 5.1. Воспроизведение AVI- или MPG-файлов средствами MCI

```
// подключаем основные системные файлы
#include <windows.h>
```

```
#include <vfw.h>
#include "resource.h"
// глобальные переменные
HINSTANCE hInst = NULL;     // дескриптор программы
HWND hwndAVI = NULL; // дескриптор окна вывода данных
BOOL bPause = FALSE; // состояние воспроизведения
BOOL bOpen = FALSE;  // состояние наличия открытого файла
LPCTSTR lpszTitle = "AVI Player"; // заголовок окна
LPCTSTR lpszAppName = "My AVI Player"; // уникальное имя программы
// главная оконная функция
LRESULT CALLBACK WindowAVIProc ( HWND hWnd, UINT uMsg, WPARAM wParam,
                                  LPARAM lParam);
// функция для открытия нового видеофайла
void OpenAVIMovie ( HWND hWnd);
// главная функция программы
int APIENTRY WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPTSTR lpCmdLine, int nCmdShow)
{
   MSG        msg;
   HWND       hWnd;
   WNDCLASSEX wc;
// определяем параметры окна программы
   wc.style          = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc    = ( WNDPROC) WindowAVIProc;
   wc.cbClsExtra     = 0;
   wc.cbWndExtra     = 0;
   wc.hInstance      = hInstance;
   wc.hIcon          = NULL;
   wc.hCursor        = LoadCursor ( NULL, IDC_ARROW);
   wc.hbrBackground = ( HBRUSH) ( COLOR_WINDOW +1);
// имя нашего меню из файла ресурсов
   wc.lpszMenuName  = "AVIPLAYER";
   wc.lpszClassName = lpszAppName;
   wc.cbSize         = sizeof ( WNDCLASSEX);
   wc.hIconSm        = NULL;
// и регистрируем собственный класс окна в системе
   if ( !RegisterClassEx ( &wc))
      return ( FALSE);
hInst = hInstance;
// создаем внешний вид окна
hWnd = CreateWindow ( lpszAppName, lpszTitle, WS_OVERLAPPEDWINDOW,
                      300, 0, 200, 0, NULL, NULL, hInstance, NULL);
// если не удалось создать окно, завершаем программу
if ( !hWnd)
      return ( FALSE);
```

```
// отображаем окно на экране
ShowWindow ( hWnd, nCmdShow);
UpdateWindow ( hWnd);
// открываем и инициализируем устройство MCI
hwndAVI = MCIWndCreate ( hWnd, hInstance, WS_CHILD | WS_VISIBLE |
            MCIWNDF_NOOPEN | MCIWNDF_NOPLAYBAR | MCIWNDF_NOTIFYSIZE |
            MCIWNDF_NOERRORDLG, NULL);
  // если открыть MCI не удалось, завершаем программу
  if ( !hwndAVI)
   {
     DestroyWindow ( hWnd);
     return ( FALSE);
   }
// запускаем цикл обработки сообщений
while ( GetMessage ( &msg, NULL, 0, 0))
   {
     TranslateMessage ( &msg);
     DispatchMessage ( &msg);
   }
return( msg.wParam);
}
// главная оконная функция нашей программы
LRESULT CALLBACK WindowAVIProc ( HWND hWnd, UINT uMsg, WPARAM wParam,
                                 LPARAM lParam)
{
switch ( uMsg)
   {
     // обрабатываем команды меню
     case WM_COMMAND:
           switch ( LOWORD ( wParam))
               {
                 // открываем файл
                 case IDM_OPEN:
                      OpenAVIMovie ( hWnd);
                   break;
                 // начинаем воспроизведение
                 case IDM_PLAY:
                      if ( hwndAVI && bOpen)
                        MCIWndPlay ( hwndAVI);
                   break;
                 // обработка паузы
                 case IDM_PAUSE:
                      bPause = !bPause;
```

```
                    // приостанавливаем воспроизведение
                    if ( bPause)
                      MCIWndPause ( hwndAVI);
                    else // продолжаем воспроизведение
                      MCIWndResume ( hwndAVI);
                  break;
               // останавливаем воспроизведение текущего файла
               case IDM_STOP:
                    MCIWndStop ( hwndAVI);
                  break;
               // переводим позицию воспроизведения в начало файла
               case IDM_RESET:
                    MCIWndHome ( hwndAVI);
                  break;
               // закрываем текущий видеофайл
               case IDM_CLOSE:
                    bOpen = FALSE;
                    bPause = FALSE;
                    MCIWndClose ( hwndAVI);
                  break;
               // завершаем работу с программой
               case IDM_EXIT:
                    // закрываем текущий видеофайл
                    MCIWndClose ( hwndAVI);
               // закрываем устройство MCI
                    MCIWndDestroy ( hwndAVI);
               // закрываем окно программы
                    DestroyWindow( hWnd);
                  break;
             }
        break;
     // завершаем работу с программой
     case WM_DESTROY :
            PostQuitMessage ( 0);
          break;
     default:
          return ( DefWindowProc ( hWnd, uMsg, wParam, lParam));
   }
   return( 0L);
}
// функция для открытия нового видеофайла
void OpenAVIMovie ( HWND hWnd)
```

```
{
   // объявляем переменные
   OPENFILENAME ofn;
   static char szFile [MAX_PATH];
   static char szFileTitle [MAX_PATH];
   // обнуляем структуру OPENFILENAME
   memset ( &ofn, 0, sizeof ( OPENFILENAME));
// инициализируем структуру OPENFILENAME
   ofn.lStructSize = sizeof ( OPENFILENAME);
   ofn.hwndOwner = hWnd;
   ofn.lpstrFilter = "AVI Files\0*.avi\0MPG Files\0*.mpg\0
                     All Files\0*.*\0\0";
   ofn.lpstrFile = szFile;
   ofn.nMaxFile = sizeof ( szFile);
   ofn.lpstrFileTitle = szFileTitle;
   ofn.nMaxFileTitle = sizeof ( szFileTitle);
   ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
   // открываем файл
   if ( GetOpenFileName ( &ofn))
      {
         // если открыт предыдущий файл, закрываем его
         if ( bOpen)
            MCIWndClose ( hwndAVI);
   bOpen = TRUE;
   if ( MCIWndOpen ( hwndAVI, ofn.lpstrFile, 0) == 0)
      {
         ShowWindow ( hwndAVI, SW_SHOW);
      }
   else // не удалось открыть выбранный файл
      {
         bOpen = FALSE;
         // выводим сообщение об ошибке
         MessageBox ( hWnd, "Не удалось открыть выбранный файл", NULL,
                     MB_ICONEXCLAMATION | MB_OK);
      }
   }
// обновляем окно просмотра
InvalidateRect ( hWnd, NULL, FALSE);
UpdateWindow ( hWnd);
}
```

Кроме основного файла, необходимо создать в редакторе ресурсов собственное меню. При этом в ваш проект будет автоматически добавлен файл

resource.h с описанием идентификаторов меню. Примеры файла и описателя ресурсов приведены в листинге 5.2 и 5.3 соответственно.

Листинг 5.2. Вид созданного редактором ресурсов файла с расширением rc

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#define APSTUDIO_READONLY_SYMBOLS
#include "afxres.h"
#undef APSTUDIO_READONLY_SYMBOLS
// Russian resources
#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_RUS)
#ifdef _WIN32
LANGUAGE LANG_RUSSIAN, SUBLANG_DEFAULT
#pragma code_page(1251)
#endif //_WIN32
// Menu
AVIPLAYER MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "Open File...",                IDM_OPEN
        MENUITEM "Close AVI File",              IDM_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "E&xit",                       40006
    END
    POPUP "&Video"
    BEGIN
        MENUITEM "Play",                        40003
        MENUITEM "Pause",                       40004
        MENUITEM "Stop",                        40005
        MENUITEM SEPARATOR
        MENUITEM "Reset",                       IDM_RESET
    END
END
#endif    // Russian resources
```

Листинг 5.3. Файл описателя ресурсов resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
```

```
// Used by Closewnd.rc
//
#define IDM_OPEN                        40001
#define 'IDM_CLOSE                      40002
#define IDM_PLAY                        40003
#define IDM_PAUSE                       40004
#define IDM_STOP                        40005
#define IDM_EXIT                        40006
#define IDM_RESET                       40007
//
// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC                     1
#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         40008
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

Не забудьте в опциях компоновщика (**Project | Settings | Link**) добавить ссылку на библиотеку Vfw.lib. После этого скомпилируйте файл и запустите. Если все было сделано правильно, в главном окне программы появится меню, состоящее из двух пунктов: **File** и **Video**. Первое меню управляет открытием и закрытием видеофайла, а также завершением работы. Оно состоит из следующих пунктов: **Open File**, **Close AVI File** и **Exit**. Второе меню содержит основные команды управления открытым файлом: **Play**, **Pause**, **Stop** и **Reset**. Кроме стандартных файлов AVI, вы сможете открывать и просматривать некоторые файлы в формате MPG.

Итак, у нас получилась простая программа для просмотра видеофайлов. Однако у нее имеется один недостаток: размеры главного окна программы автоматически не подгоняются под размер видеоизображения. Это неудобство очень легко исправить. Для этого необходимо добавить в программу обработку сообщений интерфейса MCI. Для корректировки размеров окна применяется сообщение `MCIWNDM_NOTIFYSIZE`. Оно служит для уведомления родительского окна об изменении размеров окна вывода изображения. Дополнительно следует обработать стандартное сообщение Windows, информирующее об изменении размеров окна `WM_SIZE`. Добавьте в основной файл программы (после обработки сообщения `WM_DESTROY`) обработку указанных сообщений, как показано в листинге 5.4.

Листинг 5.4. Обработка сообщений для корректировки размеров окна

```
// отрывок из функции WindowAVIProc
// завершаем работу с программой
      case WM_DESTROY:
              PostQuitMessage ( 0);
            break;
// добавляем обработку размеров окна вывода
      case WM_SIZE:
      // если окно существует и не свернуто
      if ( hwndAVI && bOpen && !IsIconic ( hWnd))
      // изменяем его размеры
           MoveWindow ( hwndAVI, 0, 0, LOWORD ( lParam),
                        HIWORD( lParam), TRUE);
            break;
// обрабатываем сообщение MCIWNDM_NOTIFYSIZE
      case MCIWNDM_NOTIFYSIZE:
       // если окно не свернуто
       if ( !IsIconic ( hWnd))
         {
           // если файл открыт
           if ( bOpen)
             {
              RECT rc;
              // устанавливаем размеры окна вывода
              GetWindowRect ( hwndAVI, &rc);
              AdjustWindowRect ( &rc, GetWindowLong ( hWnd,
                                   GWL_STYLE), TRUE);
              SetWindowPos ( hWnd, NULL, 0, 0, rc.right - rc.left,
                               rc.bottom - rc.top, SWP_NOZORDER |
                               SWP_NOACTIVATE | SWP_NOMOVE);
             }
           else
             {
             // восстанавливаем размеры окна по умолчанию
             SetWindowPos ( hWnd, NULL, 0, 0, 300, 200,
                              SWP_NOZORDER | SWP_NOACTIVATE | SWP_NOMOVE);
             }
         }
            break;
default :
            return ( DefWindowProc ( hWnd, uMsg, wParam, lParam));
```

Теперь наша программа стала более похожа на стандартный проигрыватель видеофайлов. Давайте добавим еще возможность изменения масштаба окна видеоданных. Для этого используются две макрокоманды: MCIWndGetZoom и MCIWndSetZoom. Первая получает текущее значение, а вторая позволяет установить новый масштаб. Поддерживаются три стандартных значения масштабирования: 50, 100 и 200 %. При установке значения 50 % размер окна будет уменьшен в 2 раза, а значение 200 % позволит увеличить нормальное изображение в 2 раза. Установка масштаба в 100 % возвращает нормальный размер картинки. Макрокоманда MCIWndSetZoom имеет два аргумента: первый определяет дескриптор окна вывода, а второй указывает значение желаемого масштаба. Добавьте в программу код, позволяющий увеличить изображение вдвое (листинг 5.5). Поместите его в функцию главного окна WindowAVIProc.

Листинг 5.5. Масштабирование окна видеоданных

```
// увеличиваем изображение в два раза
case IDM_ZOOMX2:
{
UINT uCurrentZoom = 0;
// получаем текущее значение
uCurrentZoom = MCIWndGetZoom ( hwndAVI);
// проверяем, имеет ли окно нормальный размер
if ( uCurrentZoom == 100)
   MCIWndSetZoom ( hwndAVI, 200); // двойной экран
}
break;
```

А теперь рассмотрим второй способ воспроизведения файлов в формате AVI.

5.2. Использование VFW

Данный метод имеет больше возможностей по сравнению с первым. Он позволяет не только воспроизводить AVI-файлы, но и получить к ним неограниченный доступ: расширенная информация о файле, работа с отдельными кадрами изображения, а также поддержка любых совместимых с Windows кодеков (компрессоров и декомпрессоров). Использование набора функций VFW позволяет не только создать полноценный видеоплеер, но и профессиональный редактор для обработки файлов в формате AVI.

Для организации воспроизведения видеофайла необходимы следующие базовые функции и макрокоманды:

- AVIFileInit. Предназначена для загрузки и инициализации набора функций VFW;

- AVIFileExit. Освобождает ресурсы системы, задействованные для VFW. Данную функцию необходимо вызвать перед завершением программы. На каждый вызов функции AVIFileInit должна быть вызвана функция AVIFileExit;
- AVIStreamOpenFromFile. Позволяет открыть новый поток для указанного файла AVI;
- AVIStreamRelease. Закрывает указанный поток для AVI-файла и освобождает используемые ресурсы;
- AVIStreamGetFrameOpen. Извлекает первый кадр из потока и подготавливает (проводит декомпрессию) его к выводу на экран;
- AVIStreamLength. Возвращает длину для указанного потока в выборках (сэмплах);
- AVIStreamEndTime. Позволяет получить конечное время для указанного потока;
- AVIStreamInfo. Читает информацию об указанном потоке. Каждый поток имеет область заголовка, в котором содержится различная информация: тип потока, размер кадра, размер одной выборки, длина потока и др.;
- AVIStreamGetFrame. Возвращает адрес восстановленного (декомпрессированного) кадра. Кадр имеет формат упакованного DIB-изображения;
- DrawDibDraw. Выводит растровое изображение на экран;
- AVIStreamRead. Позволяет прочитать видео, аудио и другие данные из указанного потока;
- DrawDibOpen. Загружает и инициализирует библиотеку для поддержки растровых изображений (DIB);
- DrawDibClose. Освобождает ресурсы, занятые библиотекой обработки растровых изображений (DIB);
- AVIStreamGetFrameClose. Освобождает ресурсы, выделенные для декомпрессии и подготовки кадров к выводу на экран.

А теперь на основе этих функций напишем собственный класс для просмотра видеоданных в формате AVI. Для удобства работы мы будем использовать средства библиотеки MFC. Это существенно упростит задачу и уменьшит размер конечного кода. Назовем наш класс CAVI и создадим его определение так, как показано в листинге 5.6.

Листинг 5.6. Файл AVI.h класс CAVI

```
// Файл AVI.h: interface for the CAVI class.
// подключим файл объявлений для библиотеки VFW
#include "vfw.h"
```

```
// сообщение для таймера
#define TIMER_UPDATE    WM_USER + 2000
// класс CAVI
class CAVI : public CWnd
{
public:
       // конструктор и деструктор по умолчанию
       CAVI ();
       virtual ~CAVI ();
// общедоступные функции класса CAVI
       // функция для загрузки AVI-файла
       int Load ( LPCTSTR lpszAVIFile);
       // функция для установки цвета фона AVI-файла
       void SetColorMask ( COLORREF color);
       // функции управления воспроизведением
       BOOL Play ();
       BOOL Stop ();
// защищенная область класса
protected:
// функции для обработки сообщений класса
// { { AFX_MSG ( CAVI)
       afx_msg void OnDestroy ();
       afx_msg void OnPaint ();
// } } AFX_MSG
       DECLARE_MESSAGE_MAP ()
// закрытая область класса
private:
       // функция для инициализации VFW
       void InitializeAVI ();
       // функция для освобождения ресурсов VFW
       void FreeAVI ();
       // функция вывода очередного кадра на экран
       void ShowFrame ( CDC* pDC);
       // функция для загрузки маски кадра
       void SetFreeScreen ( CDC* pInDC, CDC* pOutDC, int x, int y,
                            int icx, int icy);
       // функция обратного вызова для обработки событий таймера
       static void CALLBACK UpdatePlay ( HWND hWnd, UINT uMsg,
                                         UINT uEvent, DWORD dwTime);
       // переменные для хранения растровых изображений
       CBitmap m_BMPBGR;
       CBitmap* m_BMPTEMP;
       // переменная для хранения главного контекста вывода
       CDC m_DC;
```

```
    // переменная для хранения цвета маски
    COLORREF m_ColorMask;
    // переменная для хранения растрового изображения
    HDRAWDIB m_DIB;
    // переменные для хранения размеров изображения
    int m_iHeight;
    int m_iWidth;
    int m_iY;
    int m_iX;
    // переменная для хранения времени таймера
    UINT m_uTimer;
    // переменная для хранения полного числа фреймов в AVI-файле
    LONG m_CountFrame;
    // переменная для хранения текущего фрейма в AVI-файле
    UINT m_uCurFrame;
    // переменная для хранения текущего состояния воспроизведения
    BOOL m_bPlay;
    // указатель на открытый поток видео в AVI-файле
    PAVISTREAM m_pStream;
    // переменная для хранения указателя на фрейм потока видео
    PGETFRAME m_pFrame;
    // информационная структура для видеопотока
    AVISTREAMINFO AVIinfo;
};
```

Теперь напишем файл реализации класса `CAVI` так, как показано в листинге 5.7.

Листинг 5.7. Файл AVI.cpp класс `CAVI`

```
// Файл AVI.cpp: implementation of the CAVI class.
#include "stdafx.h"
#include "Video.h"
#include "AVI.h"

// конструктор класса
CAVI::CAVI ()
{
   // инициализируем переменные класса
   InitializeAVI ();
   // инициализируем VFW
   AVIFileInit ();
   // инициализируем библиотеку поддержки DIB
   m_DIB = DrawDibOpen ();
}
```

```
// деструктор класса
CAVI :: ~CAVI ()
{
    // освобождаем библиотеку DIB
    if (m_DIB) DrawDibClose ( m_DIB);
    // освобождаем ресурсы VFW
    FreeAVI ();
    // выгружаем библиотеку VFW
    AVIFileExit ();
}
// карта сообщений класса CAVI
BEGIN_MESSAGE_MAP ( CAVI, CWnd)
    ON_WM_DESTROY ()
    ON_WM_PAINT ()
END_MESSAGE_MAP ()
// функция инициализации
void CAVI :: InitializeAVI ()
{
    m_pStream   = NULL;
    m_pFrame    = NULL;
    m_bPlay     = FALSE;
    m_BMPTEMP   = NULL;
    m_uCurFrame = 0;
    m_iX        = 0;
    m_iY        = 0;
}
// функция освобождения ресурсов
void CAVI :: FreeAVI ()
{
    // если идет процесс воспроизведения, останавливаем его
    if ( m_bPlay) Stop ();
    // освобождаем ресурсы поддержки кадров
    if ( m_pFrame)
     {
       AVIStreamGetFrameClose ( m_pFrame);
       m_pFrame = NULL;
     }
    // освобождаем ресурсы видеопотока
    if ( m_pStream) AVIStreamRelease ( m_pStream);
    // удаляем объект изображения
    if ( m_BMPBGR.m_hObject) m_BMPBGR.DeleteObject ();
    // освобождаем главный контекст вывода
    if ( m_DC.m_hDC) m_DC.DeleteDC ();
}
```

```
// функция вывода очередного кадра на экран
void CAVI :: ShowFrame ( CDC* pDC)
{
    LPBITMAPINFOHEADER lpBitmap;
    CBitmap BMP, *tempBMP;
    RECT rect;
    int H = 0, W = 0;
    CDC dc;
    // получаем размеры клиентской области окна
    GetClientRect ( &rect);
    // вычисляем высоту и ширину клиентской области окна
    H = rect.bottom - rect.top;
    W = rect.right - rect.left;
    // получаем из видеопотока очередной фрейм
    lpBitmap = (LPBITMAPINFOHEADER) AVIStreamGetFrame ( m_pFrame,
                                                     (LONG) m_uCurFrame);
    // создаем на его основе совместимый контекст изображения
    if ( lpBitmap)
     {
       dc.CreateCompatibleDC ( pDC);
       BMP.CreateCompatibleBitmap ( pDC, W, H);
       tempBMP = dc.SelectObject ( &BMP);
       // копируем полученное изображение в основной контекст
       dc.BitBlt ( 0, 0, W, H, &m_DC, 0, 0, SRCCOPY);
       // выводим изображение на экран
       DrawDibDraw ( m_DIB, dc.GetSafeHdc (), rect.left + m_iX,
                     rect.top + m_iY, m_iWidth, m_iHeight, lpBitmap,
                     NULL, 0, 0, -1, -1, 0);
       // обрабатываем маску
       SetFreeScreen ( &dc, pDC, rect.left, rect.top, W, H);
       // выбираем объект
       dc.SelectObject ( tempBMP);
     }
}
// функция обработки маски кадра
void CAVI :: SetFreeScreen ( CDC* pInDC, CDC* pOutDC, int x, int y,
                             int icx, int icy)
{
    CBitmap BMP, *tempBMP;
    CBitmap copyBMP, *tempCopyBMP;
    CDC dc, copyDC;
    // создаем совместимые контексты для изображений
    dc.CreateCompatibleDC ( pInDC);
    BMP.CreateBitmap ( icx, icy, 1, 1, NULL);
```

```
    tempBMP = dc.SelectObject ( &BMP);
    copyDC.CreateCompatibleDC ( pOutDC);
    copyBMP.CreateCompatibleBitmap ( pOutDC, icx, icy);
    tempCopyBMP = copyDC.SelectObject ( &copyBMP);
    // копируем данные из одного контекста в другой
    copyDC.BitBlt ( 0, 0, icx, icy, &m_DC, 0, 0, SRCCOPY);
    // устанавливаем фон изображения
    pInDC->SetBkColor ( m_ColorMask);
    dc.BitBlt ( m_iX, m_iY, m_iWidth, m_iHeight, pInDC, m_iX, m_iY,
                SRCCOPY);
    // устанавливаем цвет фона и текста для исходного контекста
    pInDC->SetBkColor ( RGB ( 0, 0, 0));
    pInDC->SetTextColor ( RGB ( 255, 255, 255));
    pInDC->BitBlt ( m_iX, m_iY, icx, icy, &dc, m_iX, m_iY, SRCAND);
    // устанавливаем цвет для контекста маски
    copyDC.SetBkColor ( RGB ( 255, 255, 255));
    copyDC.SetTextColor ( RGB ( 0, 0, 0));
    copyDC.BitBlt ( m_iX, m_iY, m_iWidth, m_iHeight, &dc, m_iX, m_iY,
                    SRCAND);
    copyDC.BitBlt ( 0, 0, icx, icy, pInDC, 0, 0, SRCPAINT);
    // копируем данные маски в выходной контекст
    pOutDC->BitBlt ( x, y, icx, icy, &copyDC, 0, 0, SRCCOPY);
// выбираем полученные объекты изображений
    copyDC.SelectObject ( tempCopyBMP);
    dc.SelectObject ( tempBMP);
}
// функция обратного вызова для обработки событий таймера
void CALLBACK CAVI :: UpdatePlay ( HWND hWnd, UINT, UINT, DWORD)
{
    CAVI* pAVI = ( CAVI*) CWnd :: FromHandle ( hWnd);
    // если указатель пустой, выходим
    if ( pAVI == NULL) return;
    // если достигнут конец видеоданных, останавливаем игру
    if ( ++ pAVI->m_uCurFrame >= ( UINT) pAVI->m_CountFrame)
     {
       // обнуляем номер текущего фрейма
       pAVI->m_uCurFrame = 0;
       pAVI->Stop ();
       return;
     }
    // выводим следующий кадр изображения
    pAVI->Invalidate ();
}
```

```
// обработчик сообщения WM_DESTROY
void CAVI :: OnDestroy ()
{
    Stop ();
}
// обработчик сообщения WM_PAINT
void CAVI :: OnPaint ()
{
    // создаем контекст для рисования и выводим очередной кадр
    CPaintDC dc ( this);
    if ( m_pStream) ShowFrame ( &dc);

// функция для загрузки файла в формате AVI
int CAVI :: Load ( LPCTSTR lpszAVIFile)
{
    CClientDC dc ( this);
    RECT rect;
    LONG lLenghtAVI = 0L;
    int x = 0, y = 0;
    int iCenterX = 0, iCenterY = 0;
    int H = 0, W = 0;
    // обнуляем поля структуры AVISTREAMINFO
    memset ( &AVIinfo, 0, sizeof ( AVISTREAMINFO));
    // если видеопоток открыт, закрываем его
    if ( m_pStream)
     {
       FreeAVI ();
       InitializeAVI ();
     }
    // получаем размеры клиентской области окна
    GetClientRect ( &rect);
    // вычисляем высоту и ширину клиентской области окна
    H = rect.bottom - rect.top;
    W = rect.right - rect.left;
    // открываем видеопоток
    if ( AVIStreamOpenFromFile ( &m_pStream, lpszAVIFile,
         streamtypeVIDEO, 0, OF_READ | OF_SHARE_EXCLUSIVE, NULL))
     {
       m_pStream = NULL;
       return -1L; // не удалось открыть поток
     }
    // определяем полное число фреймов в потоке
    m_CountFrame = AVIStreamLength ( m_pStream);
```

```
    // подготавливаем обработчик фреймов
    m_pFrame = AVIStreamGetFrameOpen ( m_pStream, NULL);
    // получаем время конца потока
    lLenghtAVI = AVIStreamEndTime ( m_pStream);
    // вычисляем время смены кадров
    m_uTimer = ( UINT) ( lLenghtAVI / m_CountFrame);
    // получаем информацию о потоке
    AVIStreamInfo ( m_pStream, &AVIinfo, sizeof ( AVISTREAMINFO));
    // вычисляем размер видеоизображения
    x = AVIinfo.rcFrame.right - AVIinfo.rcFrame.left;
    y = AVIinfo.rcFrame.bottom - AVIinfo.rcFrame.top;
    // вычисляем координаты для центровки изображения
    iCenterX = ( ( W > x) ? ( ( W - x) / 2) : 0);
    iCenterY = ( ( H > y) ? ( ( H - y)) / 2 : 0);
    // сохраняем полученные значения
    m_iX = iCenterX;
    m_iY = iCenterY;
    m_iWidth = x;
    m_iHeight = y;
    // создаем совместимый контекст для вывода
    m_DC.CreateCompatibleDC ( &dc);
    m_BMPBGR.CreateCompatibleBitmap ( &dc, W, H);
    m_BMPTEMP = m_DC.SelectObject ( &m_BMPBGR);
    // закрашиваем клиентскую область стандартным цветом
    m_DC.FillSolidRect ( &rect, ::GetSysColor ( COLOR_WINDOW));
}
// функция для установки цвета маски
void CAVI :: SetColorMask ( COLORREF color)
{
    m_ColorMask = color;
}
// функция для запуска процесса воспроизведения
BOOL CAVI :: Play ()
{
    // если воспроизведение уже идет, выходим
    if ( m_bPlay)
     {
       m_uCurFrame = 0;
       return FALSE;
     }
    // выводим на экран первый кадр
    Invalidate ();
    // запускаем таймер
    ::SetTimer ( GetSafeHwnd (), TIMER_UPDATE, m_uTimer,
                 ( TIMERPROC) UpdatePlay);
```

```
    // устанавливаем состояние воспроизведения
    m_bPlay = TRUE;
    return TRUE;
}
// функция для остановки текущего воспроизведения
BOOL CAVI :: Stop ()
{
    // если воспроизведение уже остановлено, выходим
    if ( !m_bPlay) return FALSE;
    // останавливаем таймер
    ::KillTimer ( GetSafeHwnd (), TIMER_UPDATE);
    // устанавливаем состояние воспроизведения
    m_bPlay = FALSE;
    return TRUE;
}
```

Теперь, когда класс для воспроизведения AVI-файлов готов, создадим тестовое приложение для проверки его работы. Создайте новый проект с поддержкой библиотеки MFC (MFC AppWizard (exe)). Введите название проекта, например Video. В первом окне мастера выберите диалоговое приложение (**Dialog based**). В следующем окне мастера оставьте установленным только флаг **3D controls**. Далее нажимайте кнопки **Next** и **Finish**, пока мастер не создаст каркас программы. В редакторе ресурсов добавьте на ваше диалоговое окно кнопки **Load File**, **Play**, **Stop** и удалите кнопку **OK**. В качестве окна для вывода изображения используйте элемент управления **Static Text**. Присвойте ему уникальный идентификатор, например, IDC_VIDEO. Именно в этой области и будет осуществляться вывод видеоданных. Кнопка **Load File** позволит выбрать AVI-файл на диске и загрузить его в программу.

Скопируйте файлы класса CAVI в папку вашего тестового проекта. В файл объявлений для диалогового окна (например, VideoDlg.h) запишите ссылку на класс CAVI, как показано в листинге 5.8.

Листинг 5.8. Файл VideoDlg.h

```
// перед объявлением класса добавьте ссылку на файл CAVI.H
#include "AVI.h"
// далее следует каркас класса CVideoDlg dialog
class CVideoDlg : public CDialog
{
// Construction
public:
      CvideoDlg ( CWnd* pParent = NULL); // standard constructor
```

```
// Dialog Data
      // { { AFX_DATA ( CvideoDlg)
      enum { IDD = IDD_VIDEO_DIALOG };
          // NOTE: the ClassWizard will add data members here
      // } } AFX_DATA
      // ClassWizard generated virtual function overrides
      // { { AFX_VIRTUAL ( CvideoDlg)
      protected:
      // DDX/DDV support
      virtual void DoDataExchange ( CDataExchange* pDX);
      // } } AFX_VIRTUAL
// Implementation
protected:
      HICON m_hIcon;
      // Generated message map functions
      // { { AFX_MSG ( CvideoDlg)
      virtual BOOL OnInitDialog ();
      afx_msg void OnPaint ();
      afx_msg HCURSOR OnQueryDragIcon ();
      afx_msg void OnPlay ();
       afx_msg void OnStop ();
       afx_msg void OnOpen ();
       // } } AFX_MSG
       DECLARE_MESSAGE_MAP ()
private:
       // объявим наш класс CAVI
       CAVI m_AVI;
};
```

Файл VideoDlg.cpp генерируется мастером автоматически и имеет стандартный для диалогового приложения вид. Полный листинг этого файла можно найти на диске, прилагаемом к книге.

Добавьте в функцию инициализации диалога (в файле VideoDlg.cpp) код, как показано в листинге 5.9.

Листинг 5.9. Функция `OnInitDialog()`

```
BOOL CvideoDlg :: OnInitDialog ()
{
       Cdialog :: OnInitDialog ();
       // Set the icon for this dialog.  The framework does this
       // automatically when the application's main window
       // is not a dialog
```

```
        SetIcon ( m_hIcon, TRUE); // Set big icon
        SetIcon ( m_hIcon, FALSE); // Set small icon
        // TODO: Add extra initialization here
        // добавляем обработку событий через класс CWnd
        m_AVI.SubclassDlgItem ( IDC_VIDEO, this);
        return TRUE; // return TRUE  unless you set the focus to a control
}
```

После этого следует написать код для обработки нажатий кнопок **Play**, **Stop** и **Load File** так, как это сделано в листинге 5.10.

Листинг 5.10. Обработка нажатий управляющих кнопок

```
// функция для открытия и загрузки файла AVI
void CvideoDlg :: OnOpen ()
{
    CFileDialog dlg ( TRUE, NULL, "*.avi", OFN_HIDEREADONLY |
               OFN_FILEMUSTEXIST, "AVI Files ( *.avi)|*.avi ||", this);
    // выводим на экран диалоговое окно для открытия файлов
    if ( IDOK == dlg.DoModal ())
     {
      // получаем имя выбранного файла
      m_AVI.Load (dlg.GetPathName ());
      // устанавливаем цвет маски AVI-файла, например черный
      m_AVI.SetColorMask ( RGB ( 0, 0, 0));
     }
}
// обработка нажатия кнопки Play
void CvideoDlg :: OnPlay ()
{
    m_AVI.Play ();
}
// обработка нажатия кнопки Stop
void CvideoDlg :: OnStop ()
{
    m_AVI.Stop ();
}
```

Функция `SetColorMask` позволяет установить цвет маски изображения. Делается это для того, чтобы фон окна вывода совпадал с фоном видеопотока. Если для вас это не принципиально, данную функцию можно не вызывать. Скомпилируйте программу и проверьте ее в работе. Сразу замечу, что просматривать можно только стандартные AVI-файлы. Если же вы захотите ра-

ботать с любыми AVI-файлами, в класс CAVI следует добавить набор функций для декомпрессии видеоданных с использованием различных кодеков, установленных в системе. Как правило, для этого достаточно всего несколько базовых функций и макрокоманд:

- ICDecompressOpen. Открывает декомпрессор, совместимый с указанным форматом;
- ICDecompress. Производит декомпрессию одного фрейма;
- ICClose. Служит для закрытия компрессора и декомпрессора.

Программирование функций компрессии AVI-файлов выходит за рамки данной книги и здесь не рассматривается. Если вас интересует данная тема, ознакомьтесь с документацией, предоставляемой фирмой Microsoft.

ГЛАВА 6

Звуковая карта

Появление звуковых карт по праву может считаться началом создания полноценных мультимедийных компьютеров. Если раньше для извлечения звуков использовался системный динамик (спикер), который звучал на уровне "пустой консервной банки", то сейчас звуковой модуль представляет собой сложнейшее аналого-цифровое устройство. В состав современных звуковых карт непременно входят следующие компоненты:

- DSP (Digital Sound Processor) — звуковой процессор;
- Mixer — микшер;
- MIDI (Musical Instrument Digital Interface) интерфейс.

Каждый из этих компонентов имеет собственные порты управления и отвечает за конкретные задачи. К сожалению, звуковые карты не имеют жестко закрепленных портов ввода-вывода, что создает определенные проблемы не только программистам, но и настройщикам компьютерного оборудования. В ранних версиях Windows приходилось вручную устанавливать параметры звуковой платы, исходя из общей конфигурации оборудования. Для этого в файл autoexec.bat добавлялась строка следующего вида:

```
SET BLASTER=A220 I5 D1 [ H5 M220 P330 ]
```

Здесь:

- `A` — номер порта ввода-вывода.
- `I` — номер выделенного прерывания.
- `D` — номер выбранного 8-битного канала DMA (Direct Memory Access — прямой доступ к памяти).
- `H` — номер выбранного 16-битного канала DMA.
- `M` — номер порта ввода-вывода для микшера.

- P — номер порта ввода-вывода для MPU-401 (это один из режимов работы интерфейса MIDI).

Весь рассматриваемый в этой главе материал ориентирован на использование перечисленных ниже типов звуковых карт:

- Sound Blaster версии 1.5 (SB 1.5);
- Sound Blaster версии 2.0 (SB 2.0);
- Sound Blaster версии 2.0 с подключением CD (SB 2.0CD);
- Sound Blaster Pro (SBPRO);
- Sound Blaster 16 (SB 16);
- Sound Blaster 16 с расширенным сигнальным процессором (ASP).

Еще раз повторю, что программирование звуковых карт несколько сложнее других устройств, поскольку нет единого стандарта, а также не предусмотрена универсальная поддержка через BIOS, тем не менее, я приведу несколько специфических функций BIOS для поддержки звуковых плат.

В общем, как и ранее, представим весь материал в виде нескольких общих тем:

1. Использование функций BIOS.
2. Использование аппаратных портов.
3. Использование интерфейса Win32 API.

6.1. Использование функций BIOS

Поскольку полноценная поддержка звуковых плат в BIOS попросту отсутствует, разберем несколько специфических функций, использующих (далеко не везде) возможности драйвера SBFM (Sound Blaster Frequency Modulation). Для его работы используется прерывание `int 80h`. Приведем список основных функций:

- `0000h` — получить версию;
- `0001h` — установить адрес байта состояния;
- `0002h` — загрузить таблицу инструментов;
- `0003h` — установить частоту системных часов;
- `0004h` — установить частоту для драйвера;
- `0005h` — выполнить транспонирование;
- `0006h` — начать воспроизведение;
- `0007h` — остановить воспроизведение;
- `0008h` — выполнить сброс драйвера;

- 0009h — приостановить воспроизведение (пауза);
- 000Ah — продолжить воспроизведение после паузы.

6.1.1. Функция *0000h*

Функция позволяет получить текущую версию драйвера SBFM.

Использование:

1. В регистр BX следует поместить код функции 0000h.
2. Вызвать прерывание int 80h.

Выход:

После выполнения функции в регистр AX будет помещен номер версии.

6.1.2. Функция *0001h*

Эта функция позволяет установить адрес байта состояния, куда будет записываться результат выполненной операции.

Использование:

1. В регистр BX следует поместить код функции 0001h.
2. В DX:AX нужно записать байт состояния.
3. Вызвать прерывание int 80h.

Выход:

Выходные значения не определены.

6.1.3. Функция *0002h*

Функция позволяет загрузить новую таблицу инструментов.

Использование:

1. В регистр BX следует поместить код функции 0002h.
2. В регистр CX следует записать количество инструментов в таблице.
3. В DX:AX нужно поместить указатель на таблицу инструментов.
4. Вызвать прерывание int 80h.

Выход:

Выходные значения не определены.

6.1.4. Функция *0003h*

Данная функция позволяет установить частоту системных часов. Стандартное значение частоты равно 18,2 Гц.

Использование:

1. В регистр BX следует поместить код функции 0003h.
2. В AX следует записать значение делителя частоты. Частота генератора равна 1 193 180 Гц. Если разделить это значение на желаемую частоту (в герцах), мы получим значение делителя частоты. Для восстановления стандартного значения 18,2 Гц следует записать в этот регистр FFFFh.
3. Вызвать прерывание int 80h.

Выход:

Выходные значения не определены. Чтобы восстановить стандартную частоту, можно, например, использовать код, представленный в листинге 6.1.

Листинг 6.1. Использование функции 0003h

```
xor BX, BX     ; обнуляем регистр
mov BX, 3      ; записываем код функции 0003h
mov AX, 0FFFFh; восстанавливаем 18,2 Гц
int 80h        ; вызываем прерывание
```

6.1.5. Функция *0004h*

Функция позволяет установить значение частоты для драйвера. По умолчанию используется значение 96 Гц.

Использование:

1. В регистр BX следует поместить код функции 0004h.
2. В регистр AX следует записать значение делителя частоты. Определяется так же, как и в функции 0003h.
3. Вызвать прерывание int 80h.

Выход:

Выходные значения не определены.

6.1.6. Функция *0005h*

Функция позволяет выполнить транспонирование на указанное число полутонов.

Использование:

1. В регистр BX следует поместить код функции 0005h.
2. В регистр AX следует записать количество полутонов.
3. Вызвать прерывание int 80h.

Выход:

Выходные значения не определены. Например, чтобы транспонировать музыку на один тон вверх, можно сделать так, как показано в листинге 6.2.

Листинг 6.2. Использование функции 0005h

```
xor BX, BX     ; обнуляем регистр
mov BX, 5      ; записываем код функции 0005h
mov AX, 2      ; два полутона составляют один тон
int 80h        ; вызываем прерывание
```

6.1.7. Функция *0006h*

Данная функция предназначена для запуска процесса воспроизведения.

Использование:

1. В регистр BX следует поместить код функции 0006h.
2. В DX:AX следует поместить указатель на блок данных с музыкой.
3. Вызвать прерывание int 80h.

Выход:

В случае успешного завершения в регистр AX будет помещен результат: 0000h — операция успешно выполнена, 0001h — идет процесс воспроизведения.

6.1.8. Функция *0007h*

Эта функция позволяет остановить воспроизведение музыки.

Использование:

1. В регистр BX следует поместить код функции 0007h.
2. Вызвать прерывание int 80h.

Выход:

В случае успешного завершения в регистр AX будет помещен результат: 0000h — операция успешно выполнена, 0001h — музыка не играет.

6.1.9. Функция *0008h*

Функция позволяет выполнить сброс драйвера и восстановить используемые настройки по умолчанию.

Использование:

1. В регистр BX следует поместить код функции 0008h.
2. Вызвать прерывание int 80h.

Выход:

В случае успешного завершения в регистр AX будет помещен результат: 0000h — операция успешно выполнена, 0001h — музыка не играет.

6.1.10. Функция *0009h*

Эта функция позволяет временно приостановить воспроизведение музыки.

Использование:

1. В регистр BX следует поместить код функции 0009h.
2. Вызвать прерывание int 80h.

Выход:

В случае успешного завершения в регистр AX будет помещен результат: 0000h — операция успешно выполнена, 0001h — музыка не играет. Например, чтобы сделать паузу, можно использовать текст листинга 6.3.

Листинг 6.3. Использование функции 0009h

```
xor BX, BX    ; обнуляем регистр
xor AX, AX    ; обнуляем регистр
mov BX, 9     ; записываем код функции 0009h
int 80h       ; вызываем прерывание
cmp AX, 1     ; если музыка не играет,
je  NO_PLAY   ; передаем управление обработчику NO_PLAY
```

6.1.11. Функция *000Ah*

Функция позволяет продолжить воспроизведение, остановленное функцией 0009h.

Использование:

1. В регистр BX следует поместить код функции 000Ah.
2. Вызвать прерывание int 80h.

Выход:

В случае успешного завершения в регистр AX будет помещен результат: 0000h — операция успешно выполнена, 0001h — воспроизведение не было приостановлено.

* * *

Вот и все функции BIOS, с которыми мне хотелось вас познакомить. Не думаю, что эти сведения будут всегда полезны на практике, но помогут вам в дальнейшем (если появится полноценная поддержка BIOS) легко разобраться с новыми возможностями.

6.2. Использование портов

Поскольку современная звуковая плата состоит из нескольких модулей, мы рассмотрим программирование каждого из них в отдельности. Сегодня наиболее популярны у пользователей звуковые платы фирмы Creative. Благодаря современным технологиям, этой фирме первой удалось выпустить на рынок относительно недорогие модели звуковых плат, поддерживающие не только высококачественный звук, но и потрясающие возможности MIDI для профессиональной работы с музыкой.

Основой звуковых плат является звуковой процессор (DSP), позволяющий записывать и воспроизводить звуковой сигнал с различных источников: цифровых и аналоговых. Как правило, он поддерживает 8-ми и 16-разрядный (и более) цифровой звук, работает в моно- и стереорежимах, дуплексном режиме (воспроизведение и запись сигнала одновременно). Кроме этого, полностью реализована декомпрессия (ADPCM — Adaptive Differential Pulse Code Modulation) звука в реальном времени.

Встроенный модуль микшера управляет уровнем громкости доступных каналов, позволяет выбирать активный канал для записи и воспроизведения звука. Современные микшеры поддерживают достаточное количество микрофонных, цифровых и линейных каналов.

Интерфейс MIDI интересен в первую очередь музыкантам, поскольку позволяет писать на компьютере полноценную партитуру для своих произведений, а также подключать внешние устройства (например, синтезатор). Правда, чтобы подключить внешний музыкальный инструмент или какой-либо модуль, понадобится дополнительный переходник. Это необходимо для правильной развязки сигналов по питанию. Думаю, музыкантам это известно и так, а нам вряд ли пригодится. Замечу только, что интерфейс MIDI использует общий разъем с джойстиком (15 контактов).

Итак, рассмотрим программирование основных модулей звуковой платы:

1. Цифровой процессор (DSP).
2. Микшер.
3. Интерфейс MIDI.

6.2.1. Цифровой процессор

Цифровой процессор выполняет основную часть обработки звукового сигнала. Для ускорения работы DSP может задействовать контроллер прямого доступа к памяти (DMA). DSP используется для следующих операций:

- запись и воспроизведение моно- или стереозвука с разрядностью 8 и 16 бит;

- управление частотой дискретизации;
- в режиме эмуляции Sound Blaster декомпрессия ADPCM по формулам 4 : 1, 3 : 1 и 2 : 1;
- поддержка для Sound Blaster 16 расширенной обработки сигнала (ASP);
- управление ЦАП спикера;
- управление режимами передачи данных посредством каналов DMA.

Для доступа к DSP используются следующие базовые порты:

- `2x6h`. Используется только для записи. Позволяет выполнить программный сброс процессора и установить заданные по умолчанию параметры;
- `2xAh`. Используется только для чтения. Позволяет читать данные из DSP;
- `2xCh`. Используется для операций записи и чтения. В режиме записи является командным и позволяет посылать DSP различные управляющие команды или данные. В режиме чтения через этот порт считывается состояние готовности DSP к приему команды;
- `2xEh`. Используется только для чтения. Позволяет проверить готовность DSP для передачи данных.

Символ `x` (икс) указывает на возможные варианты адресации портов для реального компьютера: `210h`, `220h`, `230h`, `240h`, `250h`, `260h` и `280h`. Например, если для звуковой платы выбран порт `220h`, управлять DSP можно через порт `226h`.

А теперь разберем работу с этими портами подробнее.

6.2.1.1. Порт *2x6h*

Порт позволяет сбросить DSP и инициализировать значения по умолчанию. Следует обязательно выполнять эту операцию перед началом работы с процессором. Процедура сброса должна соответствовать определенной последовательности действий:

1. В порт `2x6h` записывается число 1 и выдерживается пауза минимум в 3 мкс.
2. В порт `2x6h` записывается число 0.
3. Проверяется порт `2xEh` для подтверждения полученных данных. Если бит 7 будет установлен в 1, данные получены.
4. Опрашивается порт `2xAh`. Если значение равно `0AAh`, значит DSP успешно сброшен, иначе можно предположить, что DSP отсутствует. Время опроса должно быть не менее 100 мкс.

Рассмотрим пример кода для инициализации DSP через порт `226h` (листинг 6.4).

Листинг 6.4. Инициализация DSP

```
mov DX, 226h  ; адрес порта
mov AL, 1     ; начало инициализации
out DX, AL    ; записываем значение в порт 226h
sub AL, AL    ; устанавливаем задержку
@delay:
dec AL        ; уменьшаем значение на 1
jnz @delay    ; если не 0, повторяем
out DX, AL    ; записываем в порт значение 0
sub CX, CX    ; устанавливаем количество опросов порта 22Eh
@check:
mov DX, 22Eh  ; адрес порта
in  AL, DX    ; читаем значение из порта 22Eh
and AL, 80h   ; проверяем наличие данных
jz  @nextcheck; если бит 7 равен 0, делаем новую попытку
sub DL, 4     ; читаем данные из порта 22Ah
in  AL, DX    ; получаем значение
cmp AL, 0AAh  ; код подтверждения получен
je  @EXIT     ; выходим из процедуры
@nextcheck:
loop @check   ; повторяем попытку
```

Аналогичный пример для C++ представлен в листинге 6.5.

Листинг 6.5. Инициализация DSP в C++

```
// пример функции для инициализации DSP
bool ResetDSP ( int iBasePort)
{
   DWORD dwResult  = 0; // переменная для хранения результата
   int iTimeWait = 65535;
   // записываем в порт 1
   outPort ( iBasePort + 6, 0x01, 1);
   // задержка 1 мс
   Sleep ( 1);
   // записываем в порт 0
   outPort ( iBasePort + 6, 0x00, 1);
   // проверяем наличие данных и результат операции
   while ( -- iTimeWait > 0)
    {
     // читаем состояние порта 2xEh
     inPort ( iBasePort + 0x0E, &dwResult, 1);
```

```
        if ( ( dwResult & 0x80) == 0x01)
        {
           // читаем порт 2xAh
           for ( int i = 65000; i > 0; i --)
            {
              inPort ( iBasePort + 0x0A, &dwResult, 1);
              if ( dwResult == 0xAA) return true; // DSP найден
            }
        }
      }
    // DSP не обнаружен
    return false;
}
```

6.2.1.2. Порт *2xAh*

Этот порт используется только для чтения данных из DSP. Перед тем как прочитать данные из этого порта, следует убедиться, что бит 7 в порту статуса (`2xEh`) установлен в 1. Это гарантирует наличие данных в `2xAh`, после чего их можно прочитать. Базовый пример работы с портом `22Ah` приведен в листинге 6.6.

Листинг 6.6. Чтение данных из порта 22Ah

```
mov DX, 22Eh ; адрес порта 22Eh
@repeat:
in  AL, DX   ; читаем порт статуса
or  AL, AL   ; проверяем бит 7
jns @repeat  ; данных пока нет
mov DX, 22Ah ; адрес порта 22Ah
in  AL, DX   ; читаем данные
```

Аналогичный пример для C++ показан в листинге 6.7.

Листинг 6.7. Чтение данных из порта 22Ah в C++

```
// функция для чтения данных из DSP
unsigned char ReadDSP ( int iBasePort)
{
    DWORD dwResult  = 0; // переменная для хранения результата
    int iTimeWait = 50000;
    while ( -- iTimeWait > 0)
     {
```

```
        // читаем состояние порта 2xEh
        inPort ( iBasePort + 0x0E, &dwResult, 1);
        if ( ( dwResult & 0x80) == 0x01) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return NO_TIME;
     }
    // читаем данные из порта 22Ah
    dwResult = 0;
    inPort ( iBasePort + 0x0A, &dwResult, 1);
    return (unsigned char) dwResult;
}
```

6.2.1.3. Порт *2xCh*

В режиме записи порт позволяет записать управляющую команду DSP и дополнительные аргументы, а в режиме чтения проверяет готовность DSP к приему команды. Если бит 7 сброшен в 0, значит DSP готов к приему команд или данных. Рассмотрим пример записи команды `E1h` в порт `22Ch` (листинг 6.8).

Листинг 6.8. Запись данных в порт 22Ch

```
mov DX, 22Ch ; адрес порта 22Ch
@repeat:
in  AL, DX   ; читаем порт статуса
or  AL, AL   ; проверяем бит 7
js  @repeat  ; данных пока нет
mov AL, 0E1h ; команда проверки версии DSP
out DX, AL   ; записываем команду в порт
```

Аналогичный пример для C++ показан в листинге 6.9.

Листинг 6.9. Запись данных в порт 22Ch в C++

```
// функция для записи данных в DSP
void WriteDSP ( int iBasePort, unsigned char uValue)
{
    DWORD dwResult  = 0; // переменная для хранения результата
    int iTimeWait = 50000;
    while ( -- iTimeWait > 0)
     {
        // читаем состояние порта 2xEh
        inPort ( iBasePort + 0x0C, &dwResult, 1);
```

```
        if ( ( dwResult & 0x80) == 0x00) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return;
     }
    // записываем команду в порт 22Ch
    outPort ( iBasePort + 0x0C, uValue, 1);
}
// вызываем нашу функцию для записи команды E1h в порт
WriteDSP ( 0x220, 0xE1);
```

Процессор DSP поддерживает набор управляющих команд. Все они передаются через порт 2xCh. Многие из них имеют дополнительные аргументы. Список команд приведен в табл. 6.1.

Таблица 6.1. *Команды DSP*

Код команды	Описание
10h	Прямое воспроизведение 8-битного цифрового звука
14h	Воспроизведение 8-битного цифрового звука через канал DMA
16h	Воспроизведение 2-битного ADPCM через канал DMA
17h	Воспроизведение 2-битного ADPCM через канал DMA с пустым байтом
1Ch	Воспроизведение 8-битного цифрового звука через канал DMA с автоинициализацией
1Fh	Воспроизведение 2-битного ADPCM через канал DMA с автоинициализацией
20h	Прямое чтение 8-битного цифрового звука
24h	Чтение 8-битного цифрового звука через канал DMA
2Ch	Чтение 8-битного цифрового звука через канал DMA с автоинициализацией
30h	Чтение в режиме MIDI
31h	Чтение в режиме MIDI с использованием прерывания
34h	Режим чтения и записи для UART (поллинг)
35h	Режим чтения и записи для UART с использованием прерывания
36h	Режим чтения и записи для UART (поллинг с корректировкой по времени)
37h	Режим чтения и записи для UART с использованием прерывания и корректировкой по времени
38h	Запись в режиме MIDI

Таблица 6.1 (окончание)

Код команды	Описание
40h	Установка временной константы для передачи звуковых данных
41h	Установка частоты дискретизации для воспроизведения
42h	Установка частоты дискретизации для записи
48h	Установка размера блока данных
74h	Воспроизведение 4-битного ADPCM через канал DMA
75h	Воспроизведение 4-битного ADPCM через канал DMA с пустым байтом
76h	Воспроизведение 3-битного ADPCM через канал DMA
77h	Воспроизведение 3-битного ADPCM через канал DMA с пустым байтом
7Dh	Воспроизведение 4-битного ADPCM через канал DMA с пустым байтом и автоинициализацией
7Fh	Воспроизведение 3-битного ADPCM через канал DMA с пустым байтом и автоинициализацией
80h	Установка паузы для ЦАП ввода-вывода
90h	Ускоренное воспроизведение 8-битного цифрового звука через канал DMA с автоинициализацией
91h	Ускоренная запись 8-битного цифрового звука через канал DMA
98h	Ускоренная запись 8-битного цифрового звука через канал DMA с автоинициализацией
A0h	Установка монорежима для записи
A8h	Установка стереорежима для записи
D0h	Приостановить 8-битный ввод-вывод через DMA
D1h	Включить динамик
D3h	Выключить динамик
D4h	Продолжить 8-битный ввод-вывод через DMA
D5h	Приостановить 16-битный ввод-вывод через DMA
D6h	Продолжить 16-битный ввод-вывод через DMA
D8h	Получить состояние динамика
D9h	Завершение 16-битного ввода-вывода через DMA с автоинициализацией
DAh	Завершение 8-битного ввода-вывода через DMA с автоинициализацией
E1h	Получить номер версии DSP

Разберем наиболее часто используемые команды DSP подробнее.

Команда *10h*

Команда поддерживает воспроизведение несжатого 8-битного звука. За один раз она выводит один байт данных.

Использование:

1. Записать в порт `2xCh` значение `10h`.
2. Записать в порт байт данных.
3. Подождать определенное время и перейти к первому пункту.

Следует повторять эти действия, пока не будут переданы все байты данных.

Команда *14h*

Эта команда поддерживает воспроизведение 8-битных оцифрованных данных через установленный канал DMA. Она имеет два аргумента (младший и старший байты), определяющих полную длину проигрываемого участка данных минус единица.

Использование:

1. Установить обработчик прерывания.
2. Установить частоту дискретизации (команда `40h`).
3. Записать в порт команду `D1h`.
4. Установить размер блока данных для DMA.
5. Записать в порт значение `14h` и аргументы.
6. Проверить порт `2xEh`.
7. Записать в порт команду `D3h`.

Команда *16h*

Команда поддерживает 2-битный ADPCM через канал DMA. В отличие от предыдущей команды, она работает со сжатыми данными ADPCM. Команда имеет два аргумента (младший и старший байты), определяющих полную длину передаваемых данных в байтах минус единица. В качестве первого байта данных должен быть передан пустой байт.

Команда *17h*

Данная команда поддерживает 2-битный ADPCM через канал DMA, но первый передаваемый байт должен быть пустым для обработки ADPCM.

Команда *20h*

Команда поддерживает прямое чтение 8-битного цифрового звука из АЦП. Говоря проще, данная команда позволяет записать оцифрованный звук и сохранить его на диск.

Использование:

1. Записать в порт значение `20h`.
2. Считать из порта один байт.
3. Подождать определенное время и перейти к первому пункту.

Следует повторять эти действия, пока не будут считаны все данные.

Команда *30h*

Эта команда позволяет прочитать данные из порта MIDI.

Использование:

1. Записать в порт значение `30h`.
2. Опросить DSP на наличие данных MIDI.

Команда *38h*

Команда позволяет записать данные в порт MIDI.

Использование:

1. Записать в порт значение `38h`.
2. Записать в порт данные MIDI.

Команда *40h*

Данная команда позволяет установить значение временной константы, которая может быть высчитана следующим образом:

временная константа = 65 536 – 256 000 000 / (число каналов * частота дискретизации).

Для монорежима используется один канал, а для стерео — два.

Использование:

1. Записать в порт значение `40h`.
2. Записать в порт значение константы (используется только старший байт).

Команда *41h*

Команда позволяет установить частоту дискретизации для воспроизведения звуковых данных. Допустимые значения частоты лежат в диапазоне от 5 000 до 45 000 Гц включительно.

Использование:

1. Записать в порт значение `41h`.
2. Записать в порт старший байт частоты.
3. Записать в порт младший байт частоты.

Команда *42h*

Эта команда позволяет установить частоту дискретизации для записи звуковых данных. Допустимые значения частоты лежат в диапазоне от 5 000 до 45 000 Гц включительно.

Использование:

1. Записать в порт значение `42h`.
2. Записать в порт старший байт частоты.
3. Записать в порт младший байт частоты.

Команда *48h*

Команда позволяет установить размер одного блока для ускоренного режима передачи данных DMA. Размер блока измеряется в байтах минус единица.

Использование:

1. Записать в порт значение `48h`.
2. Записать в порт младший байт.
3. Записать в порт старший байт.

Команда *80h*

Эта команда позволяет установить продолжительность паузы, которая измеряется в выборках минус единица. С помощью данной команды можно реализовать режим тишины — полное отсутствие уровня сигнала. Команда использует текущее значение частоты дискретизации.

Использование:

1. Записать в порт значение `80h`.
2. Записать в порт младший байт.
3. Записать в порт старший байт.

Команда *A0h*

Команда позволяет установить монорежим для записи. По умолчанию используется монорежим.

Команда *A1h*

Эта команда позволяет установить стереорежим для записи.

Команда *D1h*

Команда позволяет установить связь между цифровым выходом и входом усилителя. Для завершения команды необходимо максимум 112 мс.

Команда *D3h*

Команда позволяет разорвать связь между цифровым выходом и входом усилителя. Для завершения команды необходимо максимум 220 мс.

Команда *D8h*

Команда позволяет получить текущее состояние канала связи между цифровым выходом и входом усилителя. Если связи нет, возвращается `00h`, иначе — `FFh`.

Команда *E1h*

Команда позволяет получить версию установленного DSP. После выполнения команды будут получены два байта: первый определяет старший (major) номер версии, второй содержит младший (minor) номер.

* * *

Рассмотрим несколько примеров работы с DSP. Напишем код для включения динамика (листинг 6.10).

Листинг 6.10. Включение динамика

```
mov DX, 22Ch ; адрес порта 22Ch
@repeat:
in  AL, DX   ; читаем порт статуса
or  AL, AL   ; проверяем бит 7
js  @repeat  ; данных пока нет
mov AL, 0D1h ; включаем динамик
out DX, AL   ; записываем команду в порт
```

Аналогичный пример для C++ представлен в листинге 6.11.

Листинг 6.11. Включение динамика в C++

```
// вызываем нашу функцию для записи команды D1h в порт
WriteDSP ( 0x220, 0xD1);
```

А теперь попробуем установить частоту дискретизации для воспроизведения 44 100 Гц (`0AC44h`) так, как это сделано в листинге 6.12.

Листинг 6.12. Установка новой частоты дискретизации

```
writeDSP proc near
mov DX, 22Ch ; адрес порта 22Ch
@repeat:
in  AL, DX   ; читаем порт статуса
```

```
or  AL, AL   ; проверяем бит 7
js  @repeat  ; данных пока нет
mov AL, BL   ; записываем данные
out DX, AL   ; записываем команду в порт
writeDSP endp
xor BX, BX   ; обнуляем регистр
mov BL, 41h  ; выбираем команду установки новой частоты
call writeDSP; записываем команду в порт
mov BL, 0ACh ; сначала пишем старший байт частоты
call writeDSP; записываем значение в порт
mov BL, 44h  ; потом пишем младший байт частоты
call writeDSP; записываем значение в порт
```

Аналогичный пример для C++ показан в листинге 6.13.

Листинг 6.13. Установка новой частоты дискретизации в C++

```
// используем нашу функцию для записи значений в порт DSP
WriteDSP ( 0x220, 0x41); // код команды 41h
WriteDSP ( 0x220, 0xAC); // старший байт частоты
WriteDSP ( 0x220, 0x44); // младший байт частоты
```

Для установки DSP в 16-битный режим воспроизведения и передачи блока данных размером 16 Кб можно использовать код из листинга 6.14.

Листинг 6.14. Установка 16-битного режима

```
writeDSP proc near
mov DX, 22Ch ; адрес порта 22Ch
@repeat:
in  AL, DX   ; читаем порт статуса
or  AL, AL   ; проверяем бит 7
js  @repeat  ; данных пока нет
mov AL, BL   ; записываем данные
out DX, AL   ; записываем команду в порт
writeDSP endp
xor BX, BX   ; обнуляем регистр
mov BL, 0B0h ; воспроизведение 16-ти бит
call writeDSP; записываем команду в порт
mov BL, 30h  ; стереорежим
call writeDSP; записываем команду в порт
mov BL, 0FFh ; младший байт длины блока
```

```
call writeDSP; записываем команду в порт
mov BL, 03h  ; старший байт длины блока
call writeDSP; записываем команду в порт
```

Аналогичный пример для C++ показан в листинге 6.15.

Листинг 6.15. Установка 16-битного режима в C++

```
// используем нашу функцию для записи значений в порт DSP
WriteDSP ( 0x220, 0xB0); // воспроизведение 16-ти бит
WriteDSP ( 0x220, 0x30); // стереорежим
WriteDSP ( 0x220, 0xFF); // младший байт длины блока
WriteDSP ( 0x220, 0x03); // старший байт длины блока
```

Длина блока данных в обоих примерах рассчитана умножением числа 16 на 1 024 минус один. Результирующее значение равно 16383 или 3FFFh в шестнадцатеричном исчислении. На этом завершим работу с DSP и рассмотрим, как программируется микшер звуковой карты.

6.2.2. Микшер

Микшер, как уже было сказано ранее, предназначен для коммутации каналов, а также для управления громкостью звука этих каналов. Для поддержки работы с микшером предназначены следующие порты:

- 2x4h — адресный порт. Используется только для записи. Позволяет выбрать номер управляющего регистра;
- 2x5h — порт данных. Используется для чтения и записи. Позволяет передавать и получать данные для указанного регистра.

Для работы с микшером необходимо записать в порт 2x4h номер управляющего регистра. После этого можно записывать или считывать данные из порта 2x5h. При изменении отдельных битов управляющего регистра вначале следует прочитать его значение, а после этого изменить нужный разряд и записать значение в порт. Важно соблюдать это правило для совместимости с последующими расширениями. Список управляющих регистров приведен в табл. 6.2.

Приведем краткое описание таблицы.

- Регистр 00h предназначен для сброса микшера. После сброса все регистры микшера будут установлены по умолчанию. Для выполнения сброса достаточно записать любое значение (размером в байт) в этот регистр.

Таблица 6.2. *Список регистров микшера*

Код регистра	Описание							
	7	6	5	4	3	2	1	0
00h	Сброс микшера							
04h	Громкость ЦАП в левом канале				Громкость ЦАП в правом канале			
0Ah	Резерв					Громкость микрофона		
22h	Общая громкость в левом канале				Общая громкость в правом канале			
26h	Громкость MIDI в левом канале				Громкость MIDI в правом канале			
28h	Громкость CD в левом канале				Громкость CD в правом канале			
2Eh	Громкость линейного входа в левом канале				Громкость линейного входа в правом канале			
30h	Общая громкость в левом канале					Резерв		
31h	Общая громкость в правом канале					Резерв		
32h	Громкость ЦАП в левом канале					Резерв		
33h	Громкость ЦАП в правом канале					Резерв		
34h	Громкость MIDI в левом канале					Резерв		
35h	Громкость MIDI в правом канале					Резерв		
36h	Громкость CD в левом канале					Резерв		
37h	Громкость CD в правом канале					Резерв		
38h	Громкость линейного входа в левом канале					Резерв		
39h	Громкость линейного входа в правом канале					Резерв		
3Ah	Громкость микрофона					Резерв		
3Bh	Громкость динамика ПК		Резерв					
3Ch	Резерв			Управление выходными каналами				
				ЛЛ	ЛП	АЛ	АП	М
3Dh	Резерв	Управление входным левым каналом						
		МЛ	МП	ЛЛ	ЛП	АЛ	АП	М
3Eh	Резерв	Управление входным правым каналом						
		МЛ	МП	ЛЛ	ЛП	АЛ	АП	М
3Fh	УЛВХК		Резерв					
40h	УПВХК		Резерв					

Таблица 6.2 (окончание)

Код регистра	Описание							
	7	6	5	4	3	2	1	0
41h	УЛВК		Резерв					
42h	УПВК		Резерв					
43h	Резерв							АПУ
44h	Уровень высоких частот в левом канале				Резерв			
45h	Уровень высоких частот в правом канале				Резерв			
46h	Уровень низких частот в левом канале				Резерв			
47h	Уровень низких частот в правом канале				Резерв			

- Регистр 04h управляет уровнем громкости ЦАП в левом и правом каналах. Биты 7—4 отвечают за левый канал, а биты 3—0 — за правый. Имеются 16 уровней (от 0 до 15) громкости, где 0 соответствует значению –60 дБ, а 15 — 0 дБ с шагом в 4 дБ. По умолчанию уровень равен 12 (12 дБ).
- Регистр 0Ah управляет уровнем громкости микрофона. Используются только биты 2—0. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению –48 дБ, а 7 — 0 дБ с шагом в 6 дБ. По умолчанию уровень равен 0 (–48 дБ).
- Регистр 22h управляет общим уровнем громкости в левом и правом каналах. Биты 7—4 отвечают за левый канал, а биты 3—0 — за правый. Имеются 16 уровней (от 0 до 15) громкости, где 0 соответствует значению –60 дБ, а 15 — 0 дБ с шагом в 4 дБ. По умолчанию уровень равен 12 (12 дБ).
- Регистр 26h управляет уровнем громкости MIDI в левом и правом каналах. Биты 7—4 отвечают за левый канал, а биты 3—0 — за правый. Имеются 16 уровней (от 0 до 15) громкости, где 0 соответствует значению –60 дБ, а 15 — 0 дБ с шагом в 4 дБ. По умолчанию уровень равен 12 (12 дБ).
- Регистр 28h управляет уровнем громкости аудиодиска в левом и правом каналах. Биты 7—4 отвечают за левый канал, а биты 3—0 — за правый. Имеются 16 уровней (от 0 до 15) громкости, где 0 соответствует значению –60 дБ, а 15 — 0 дБ с шагом в 4 дБ. По умолчанию уровень равен 0 (–60 дБ).

- Регистр `2Eh` управляет уровнем громкости линейного входа в левом и правом каналах. Биты 7—4 отвечают за левый канал, а биты 3—0 — за правый. Имеются 16 уровней (от 0 до 15) громкости, где 0 соответствует значению –60 дБ, а 15 — 0 дБ с шагом в 4 дБ. По умолчанию уровень равен 0 (–60 дБ).
- Регистр `30h` управляет общим уровнем громкости в левом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 24 (–14 дБ).
- Регистр `31h` управляет общим уровнем громкости в правом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 24 (–14 дБ).
- Регистр `32h` управляет уровнем громкости ЦАП в левом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 24 (–14 дБ).
- Регистр `33h` управляет уровнем громкости ЦАП в правом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 24 (–14 дБ).
- Регистр `34h` управляет уровнем громкости MIDI в левом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 24 (–14 дБ).
- Регистр `35h` управляет уровнем громкости MIDI в правом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 24 (–14 дБ).
- Регистр `36h` управляет уровнем громкости аудиодиска в левом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 0 (–62 дБ).
- Регистр `37h` управляет уровнем громкости аудиодиска в правом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 0 (–62 дБ).
- Регистр `38h` управляет уровнем громкости линейного входа в левом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 0 (–62 дБ).

- Регистр `39h` управляет уровнем громкости линейного входа в правом канале. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 0 (–62 дБ).
- Регистр `3Ah` управляет уровнем громкости микрофона. Используются только биты 7—3. Имеются 32 уровня (от 0 до 31) громкости, где 0 соответствует значению –62 дБ, а 31 — 0 дБ с шагом в 2 дБ. По умолчанию уровень равен 0 (–62 дБ).
- Регистр `3Bh` управляет уровнем громкости динамика ПК. Используются только биты 7 и 6. Имеются 4 уровня (от 0 до 3) громкости, где 0 соответствует значению –18 дБ, а 3 — 0 дБ с шагом в 6 дБ. По умолчанию уровень равен 0 (–18 дБ).
- Регистр `3Ch` управляет выходными каналами. Используются только биты 4—0: бит 4 — линейный левый канал, 3 — линейный правый канал, 2 — аудио CD левый, 1 — аудио CD правый, 0 — микрофон. Установка бита в 0 позволяет открыть соответствующий канал. По умолчанию значения всех битов равны 1 (все закрыты).
- Регистр `3Dh` управляет левыми входными каналами. Используются только биты 6—0: бит 6 — MIDI левый (0), 5 — MIDI правый (0), 4 — линейный левый (1), 3 — линейный правый (0), 2 — аудио CD левый (1), 1 — аудио CD правый (0), 0 — микрофон (1). Значения по умолчанию указаны в круглых скобках. Установка бита в 0 позволяет открыть соответствующий канал. В монорежиме являются единственными каналами для ввода данных.
- Регистр `3Eh` управляет правыми входными каналами. Используются только биты 6—0: бит 6 — MIDI левый (0), 5 — MIDI правый (0), 4 — линейный левый (1), 3 — линейный правый (0), 2 — аудио CD левый (1), 1 — аудио CD правый (0), 0 — микрофон (1). Значения по умолчанию указаны в круглых скобках. Установка бита в 0 позволяет открыть соответствующий канал.
- Регистр `3Fh` управляет уровнем усиления в левом входном канале. Используются только биты 7 и 6. Имеются 4 уровня (от 0 до 3) усиления, где 0 соответствует значению 0 дБ, а 3 — 18 дБ с шагом в 6 дБ. По умолчанию уровень равен 0 (0 дБ).
- Регистр `40h` управляет уровнем усиления в правом входном канале. Используются только биты 7 и 6. Имеются 4 уровня (от 0 до 3) усиления, где 0 соответствует значению 0 дБ, а 3 — 18 дБ с шагом в 6 дБ. По умолчанию уровень равен 0 (0 дБ).
- Регистр `41h` управляет уровнем усиления в левом выходном канале. Используются только биты 7 и 6. Имеются 4 уровня (от 0 до 3) усиления, где 0 соответствует значению 0 дБ, а 3 — 18 дБ с шагом в 6 дБ. По умолчанию уровень равен 0 (0 дБ).

- Регистр 42h управляет уровнем усиления в правом выходном канале. Используются только биты 7 и 6. Имеются 4 уровня (от 0 до 3) усиления, где 0 соответствует значению 0 дБ, а 3 — 18 дБ с шагом в 6 дБ. По умолчанию уровень равен 0 (0 дБ).
- Регистр 43h управляет автоматической подстройкой уровня усиления микрофона. Используется только бит 0. Установка бита в 1 позволяет увеличить уровень на 20 дБ. По умолчанию бит равен 0 (0 дБ).
- Регистр 44h управляет уровнем высоких частот в левом канале. Используются только биты 7—4. Имеются 16 уровней. Значения от 0 до 7 устанавливают соответственно значения уровня от –14 дБ до 0 дБ с шагом 2 дБ. Значения от 8 до 15 устанавливают значения уровня соответственно от 0 дБ до 14 дБ с шагом 2 дБ. По умолчанию используется значение 8 (0 дБ).
- Регистр 45h управляет уровнем высоких частот в правом канале. Используются только биты 7—4. Имеются 16 уровней. Значения от 0 до 7 устанавливают значения уровня соответственно от –14 дБ до 0 дБ с шагом 2 дБ. Значения от 8 до 15 устанавливают значения уровня соответственно от 0 дБ до 14 дБ с шагом 2 дБ. По умолчанию используется значение 8 (0 дБ).
- Регистр 46h управляет уровнем низких частот в левом канале. Используются только биты 7—4. Имеется 16 уровней. Значения от 0 до 7 устанавливают соответственно значения уровня от –14 дБ до 0 дБ с шагом 2 дБ. Значения от 8 до 15 устанавливают значения уровня соответственно от 0 дБ до 14 дБ с шагом 2 дБ. По умолчанию используется значение 8 (0 дБ).
- Регистр 47h управляет уровнем низких частот в правом канале. Используются только биты 7—4. Имеются 16 уровней. Значения от 0 до 7 устанавливают соответственно значения уровня от –14 дБ до 0 дБ с шагом 2 дБ. Значения от 8 до 15 устанавливают соответственно значения уровня от 0 дБ до 14 дБ с шагом 2 дБ. По умолчанию используется значение 8 (0 дБ).
- Регистр 80h позволяет установить номер прерывания для звуковой карты. Используются только биты с 3 по 0: бит 3 — прерывание IRQ 10, 2 бит — IRQ 7, 1 бит — IRQ 5, 0 бит — IRQ 2. Следует помнить, что в один момент времени может быть установлен только один номер прерывания. Для этого нужно записать в соответствующий бит значение 1. Хотя этот регистр и управляет общими настройками всей платы, работать с ним нужно через порты микшера.
- Регистр 81h позволяет установить номер канала DMA для звуковой карты. Как и предыдущий регистр, он управляет глобальными настройками звуковой платы, но адресуется через порты микшера. Формат данного

регистра показан в табл. 6.3. Для выбора нужного канала следует установить соответствующий бит в 1. В один момент времени может быть установлен только один из трех 16-битных и 8-битных каналов DMA. Производители звуковых плат не рекомендуют изменять значения в регистрах `80h` и `81h`, а предлагают пользоваться специальными утилитами, входящими в комплект поставки звуковой платы. Кроме того, эти регистры могут использоваться только для PnP (Plug and Play) плат.

Таблица 6.3. *Формат регистра* `81h`

Бит	7	6	5	4	3	2	1	0
Канал DMA	DMA 7 (16 бит)	DMA6 (16 бит)	DMA 5 (16 бит)	Резерв	DMA 3 (8 бит)	Резерв	DMA 1 (8 бит)	DMA 0 (8 бит)

- Регистр `82h` предназначен для получения состояния об аппаратном прерывании звуковой платы. Может применяться для работы со всеми модулями звуковой платы: DSP, микшером и интерфейсом MPU-401. Работа с регистром выполняется через порты микшера. Регистр можно только читать. Информационными являются следующие биты: 2 — интерфейс MPU-401, 1 — 16-битный канал DMA для ввода и вывода, 0 — 8-битный канал DMA для ввода и вывода. Если считываемый бит установлен в 1, значит, прерывание было вызвано соответствующим компонентом (MPU-401, DMA 16-ти бит или DMA 8-ми бит). Для подтверждения прерывания следует прочитать один из следующих портов: `2xEh` — для DMA 8-ми бит или SB-MIDI, `2xFh` — для 16-ти бит DMA, `3x0h` — для MPU-401.

Как видите, управлять микшером не очень сложно. Рассмотрим простой пример для установки новых значений уровня общей громкости для левого и правого каналов (листинг 6.16).

Листинг 6.16. Установка общего уровня громкости

```
; предположим, что микшер доступен через порты 224h и 225h
mov DX, 224h ; адресный регистр
mov AL, 30h  ; регистр 30h
out DX, AL   ; записываем значение в порт
inc DX       ; регистр данных
mov AL, 14h  ; установим уровень громкости -22 дБ для левого канала
shl AL, 3    ; используются только биты с 7 по 3
out DX, AL   ; записываем значение в порт
dec DX       ; адресный регистр
mov AL, 31h  ; регистр 31h
```

```
out DX, AL   ; записываем значение в порт
inc DX       ; регистр данных
mov AL, 14h  ; установим уровень громкости -22 дБ для правого канала
shl AL, 3    ; используются только биты с 7 по 3
out DX, AL   ; записываем значение в порт
```

Аналогичный пример для C++ представлен в листинге 6.17.

Листинг 6.17. Установка общего уровня громкости в C++

```
// функция для записи данных в DSP
void SetMasterVolume ( DWORD dwValueL, DWORD dwValueR)
{
    DWORD dwValue = 0;
    // записываем номер регистра в порт 224h
    outPort ( 0x224, 0x30, 1);
    dwValue = dwValueL << 3; // используются только биты с 7 по 3
    outPort ( 0x225, dwValue, 1); // левый канал
    // записываем номер регистра в порт 224h
    outPort ( 0x224, 0x31, 1);
    dwValue = dwValueR << 3; // используются только биты с 7 по 3
    outPort ( 0x225, dwValue, 1); // левый канал
}
// установим общий уровень громкости -22 дБ для обоих каналов
SetMasterVolume ( 20, 20);
```

А теперь попробуем определить, каким модулем было вызвано последнее прерывание (листинг 6.18).

Листинг 6.18. Определение источника прерывания

```
; предположим, что микшер доступен через порты 224h и 225h
mov DX, 224h ; адресный регистр
mov AL, 82h  ; регистр 82h
out DX, AL   ; пишем в порт
inc DX       ; выбираем порт 225h
in  AL, DX   ; читаем из него байт
cmp AL, 01h  ; DMA 8-ми бит
je  dma_8    ; переходим в процедуру dma_8 и проверяем порт 22Eh
cmp AL, 02h  ; DMA 16-ти бит
je  dma_16   ; переходим в процедуру dma_16 и проверяем порт 22Fh
cmp AL, 04h  ; MPU-401
je  mpu_401  ; переходим в процедуру mpu_401 и проверяем порт 3x0h
```

Для подтверждения прерывания следует прочитать байт из соответствующего порта (2xEh, 2xFh или 3x0h), а после этого послать команду завершения прерывания в порт программируемого контроллера прерываний. Аналогичный пример для C++ показан в листинге 6.19.

Листинг 6.19. Определение источника прерывания в C++

```
// переменная для хранения результата
DWORD dwResult = 0;
// записываем номер регистра 82h в порт 224h
outPort ( 0x224, 0x82, 1);
// читаем порт 225h
inPort ( 0x225, &dwResult, 1);
// проверяем, чем вызвано прерывание
switch ( dwResult)
{
  case 1: // DMA 8-ми бит
   // выполняем какие-либо действия
   break;
  case 2: // DMA 16-ти бит
   // выполняем какие-либо действия
   break;
  case 4: // интерфейс MPU-401
   // выполняем какие-либо действия
   break;
}
```

Вся информация об устройстве микшера приведена для модуля CT1745 фирмы Creative. Если читатель программирует более ранние версии микшера, то ему следует полагаться на информацию, представленную в табл. 6.4 и 6.5.

Таблица 6.4. *Список регистров микшера CT1335*

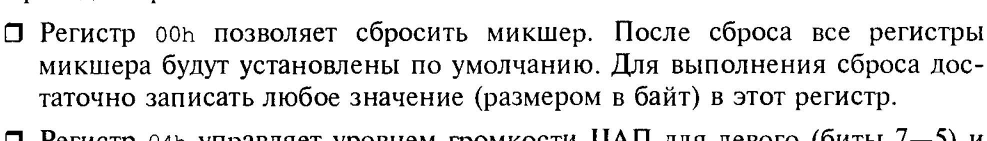
Код регистра	Описание							
	7	6	5	4	3	2	1	0
00h	Сброс микшера							
02h	Резерв				Общий уровень громкости			Резерв
06h	Резерв				Уровень громкости MIDI			Резерв
08h	Резерв				Уровень громкости аудио CD			Резерв
0Ah	Резерв					Уровень громкости ЦАП		Резерв

Приведем краткое описание табл. 6.4.

- Регистр `00h` позволяет сбросить микшер. После сброса все регистры микшера будут установлены в значения, используемые по умолчанию. Для выполнения сброса достаточно записать любое значение (размером в байт) в этот регистр.
- Регистр `02h` управляет общим уровнем громкости всех каналов. Используются только биты 3—1. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению –46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 4 (–11 дБ).
- Регистр `06h` управляет уровнем громкости MIDI. Используются только биты 3—1. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению –46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 4 (–11 дБ).
- Регистр `08h` управляет уровнем громкости аудиодиска. Используются только биты 3—1. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению –46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 0 (–46 дБ).
- Регистр `0Ah` управляет уровнем громкости ЦАП. Используются только биты 2 и 1. Имеются 4 уровня (от 0 до 3) громкости, где 0 соответствует значению –46 дБ, а 3 — 0 дБ с нелинейным шагом в 7 дБ. По умолчанию уровень равен 0 (–46 дБ).

Таблица 6.5. *Список регистров микшера CT1345*

Код регистра	Описание							
	7	6	5	4	3	2	1	0
00h	Сброс микшера							
04h	Уровень громкости ЦАП для левого канала			Резерв	Уровень громкости ЦАП для правого канала			Резерв
0Ah	Резерв					M		Резерв
0Ch	Резерв		ВХФ	Резерв	Фильтр НЧ	Входной источник		Резерв
0Eh	Резерв		ВЫХФ	Резерв			Стерео	Резерв
22h	Общий уровень в левом канале			Резерв	Общий уровень в правом канале			Резерв
26h	Уровень MIDI в левом канале			Резерв	Уровень MIDI в правом канале			Резерв
28h	Уровень CD в левом канале			Резерв	Уровень CD в правом канале			Резерв
2Eh	Линейный левый канал			Резерв	Линейный правый канал			Резерв



Приведем краткое описание табл. 6.5.

- Регистр 00h позволяет сбросить микшер. После сброса все регистры микшера будут установлены по умолчанию. Для выполнения сброса достаточно записать любое значение (размером в байт) в этот регистр.
- Регистр 04h управляет уровнем громкости ЦАП для левого (биты 7—5) и правого (биты 3—1) каналов. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению −46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 4 (–11 дБ).
- Регистр 0Ah управляет уровнем громкости микрофона (биты 2 и 1). Имеются 4 уровня (от 0 до 3) громкости, где 0 соответствует значению –46 дБ, а 3 — 0 дБ с нелинейным шагом в 7 дБ. По умолчанию уровень равен 0 (–46 дБ).
- Регистр 0Ch имеет следующее назначение: бит 5 управляет состоянием входного фильтра: 0 — включен и сигнал проходит через фильтр нижних частот, 1 — выключен и сигнал идет в обход фильтра нижних частот. По умолчанию бит 5 равен 0. Бит 3 позволяет выбрать частоту для фильтра нижних частот, если бит 5 установлен в 0. Если бит 3 равен 0, будет выбрана частота 3,2 кГц, если 1 — 8,8 кГц. По умолчанию бит 3 равен 0. Биты 2 и 1 позволяют выбрать входной источник: 0 или 2 — микрофон, 1 — аудиодиск, 3 — линейный вход.
- Регистр 0Eh имеет следующее назначение: бит 5 управляет состоянием выходного фильтра: 0 — включен и сигнал проходит через фильтр нижних частот, 1 — выключен и сигнал идет в обход фильтра нижних частот. По умолчанию бит 5 равен 0. Для стереорежима бит 5 следует установить в 1. Бит 1 управляет стереорежимом для выходного канала: 0 — моно, 1 — стерео. По умолчанию используется моно (0) режим.
- Регистр 22h управляет общим уровнем громкости для левого (биты 7—5) и правого (биты 3—1) каналов. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению –46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 4 (–11 дБ).
- Регистр 26h управляет уровнем громкости MIDI для левого (биты 7—5) и правого (биты 3—1) каналов. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению −46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 4 (–11 дБ).
- Регистр 28h управляет уровнем громкости аудиодиска для левого (биты 7—5) и правого (биты 3—1) каналов. Имеются 8 уровней (от 0 до 7) громкости, где 0 соответствует значению –46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 0 (–46 дБ).
- Регистр 2Eh управляет уровнем громкости для левого (биты 7—5) и правого (биты 3—1) линейных каналов. Имеются 8 уровней (от 0 до 7) гром-

кости, где 0 соответствует значению –46 дБ, а 7 — 0 дБ с нелинейным шагом в 4 дБ. По умолчанию уровень равен 0 (–46 дБ).

На этом можно завершить описание устройства микшера.

6.2.3. Интерфейс MIDI

Звуковая плата обычно поддерживает два основных интерфейса MIDI: MPU-401 и SB-MIDI.

В интерфейсе SB-MIDI используются те же порты ввода-вывода, что и для процессора DSP. Передача данных может выполняться в двух режимах: нормальном и UART (Universal Asynchronous Receiver/Transmitter — универсальный асинхронный режим приема и передачи данных). В нормальном режиме сначала записывается команда MIDI, а после нее данные. В режиме UART следует сразу послать DSP одну из команд разрешения работы с UART (34h, 35h, 36h или 37h). После того как DSP будет переведен в режим UART, можно читать или писать MIDI-данные. После завершения работы следует послать DSP-команду сброса, чтобы восстановить нормальный режим его работы. Для обнаружения MIDI-данных на входе используются два режима: поллинга и прерывания. В режиме поллинга при поступлении данных на вход бит 7 в порту статуса (2xEh) устанавливается в 1. Если данных нет, этот бит равен 0. Во втором режиме при поступлении данных происходит прерывание. Для перехвата прерывания пишется специальная функция. В любом из этих режимов чтение данных происходит одинаково:

1. Опрашивается порт 2xEh.
2. Если бит 7 установлен в 1, считываются данные из порта 2xAh.
3. Если бит 7 равен 0, переход к первому пункту.

В интерфейсе MPU-401 (режим UART) все данные передаются без изменений между компьютером и MIDI-устройством. Когда DSP установлен в режим UART, он не реагирует ни на какие команды, кроме сброса (00h). Интерфейс MPU-401 в режиме UART использует аппаратное прерывание и выделенные адреса ввода-вывода. Как правило, могут применяться следующие номера прерываний: 2, 5 (по умолчанию), 7 или 10. Для адресации используются базовые порты 300h или 330h (по умолчанию). При этом порты ввода-вывода могут иметь следующие адреса:. 300h и 301h или 330h и 331h.

Рассмотрим назначение портов подробнее.

- Порт 3x1h в режиме чтения является портом состояния. С помощью данного порта можно определить наличие данных в порту данных. Используются только биты 7 и 6. Бит 7 отвечает за готовность к вводу данных: 0 — данные доступны для считывания, 1 — нет данных для чтения. Бит 6 отвечает за готовность данных на выходе: 0 — интерфейс готов к приему

данных или кода команды, 1 — интерфейс не готов принять данные или код команды.

- Порт `3x1h` в режиме записи служит для посылки кода управляющей команды.
- Порт `3x0h` — порт данных. Используется для считывания и записи данных.

В листинге 6.20 показано, как следует проверять готовность интерфейса MIDI к приему данных.

Листинг 6.20. Проверка готовности интерфейса MIDI

```
IsMidi proc near
mov DX, 301h ; например, базовый регистр равен 300h
@repaet:     ; проверяем порт состояния 301h
in  AL, DX   ; читаем байт из порта
test AL, 40h ; если бит 6 равен 1
jnz @repeat  ; повторяем опрос порта
IsMidi endp
; пишем данные в порт
call IsMidi  ; проверяем, готов ли порт
mov DX, 301h ; порт команды
mov AL, bCmd ; код команды
out DX, AL   ; пишем команду в порт
call IsMidi  ; проверяем, готов ли порт
mov DX, 300h ; порт данных
mov AL, bData; можно передавать данные
out DX, AL   ; пишем данные в порт
```

Аналогичный пример на С++ показан в листинге 6.21.

Листинг 6.21. Проверка готовности интерфейса MIDI в С++

```
// пишем функцию проверки порта
bool IsReadMIDI ()
{
    DWORD dwResult  = 0; // переменная для хранения результата
    int iTimeWait = 50000;
    while ( -- iTimeWait > 0)
     {
        // читаем состояние порта 301h
        inPort ( 0x301, &dwResult, 1);
        if ( ( dwResult & 0x40) == 0x00) return true;
```

```
        // закончилось время ожидания
        if ( iTimeWait < 1) break;
     }
    return false;
}
// используем нашу функцию для записи данных в порт 300h
if ( IsReadMIDI)
  outPort ( 0x301, dwCommand, 1); // записываем команду в порт
if ( IsReadMIDI)
  outPort ( 0x300, dwData, 1); // записываем данные в порт
```

Разберем еще один пример, позволяющий определить наличие данных для считывания из порта MIDI (листинг 6.22).

Листинг 6.22. Определение наличия в порту MIDI данных

```
IsMidi proc near
mov DX, 301h ; например, базовый регистр равен 300h
@repaet:     ; проверяем порт состояния 301h
in  AL, DX   ; читаем байт из порта
test AL, 80h ; если бит 7 равен 1
jnz @repeat  ; повторяем опрос порта
IsMidi endp
; пишем данные в порт
call IsMidi  ; проверяем, есть ли данные
mov DX, 300h ; порт данных
in  AL, DX   ; читаем байт данных из порта
```

Аналогичный пример для C++ показан в листинге 6.23.

Листинг 6.23. Определение наличия в порту MIDI данных в C++

```
// пишем функцию для проверки наличия данных в порту
bool IsDataMIDI ()
{
    DWORD dwResult  = 0; // переменная для хранения результата
    int iTimeWait = 50000;
    while ( -- iTimeWait > 0)
     {
        // читаем состояние порта 301h
        inPort ( 0x301, &dwResult, 1);
        if ( ( dwResult & 0x80) == 0x00) return true;
```

```
        // закончилось время ожидания
        if ( iTimeWait < 1) break;
     }
   return false;
}
// используем нашу функцию для чтения данных из порта 300h
if ( IsDataMIDI)
  inPort ( 0x300, &dwResult, 1); // читаем данные
```

В режиме UART поддерживаются только две команды: сброса и включения режима UART. Команда сброса с кодом `FFh` позволяет выполнить сброс интерфейса MPU-401. Если команда успешно выполнена, интерфейс возвратит значение `FEh` через порт данных (`3x0h`). Команда сброса может использоваться для проверки наличия интерфейса MPU-401. Например, чтобы выполнить сброс и убедиться в наличии интерфейса, можно применить код, показанный в листинге 6.24.

Листинг 6.24. Проверка поддержки интерфейса MPU-401

```
IsMidi proc near
mov DX, 301h ; например, базовый регистр равен 300h
@repaet:     ; проверяем порт состояния 301h
in  AL, DX   ; читаем байт из порта
test AL, 40h ; если бит 6 равен 1
jnz @repeat  ; повторяем опрос порта
IsMidi endp
; основной код
call IsMidi  ; проверяем готовность MIDI
mov DX, 301h ; например, базовый регистр равен 300h
mov AL,0FFh  ; команда сброса MPU-401
out DX, AL   ; пишем в порт
sub CX, CX   ; устанавливаем значение счетчика ожидания
inc DX       ; порт 301h
@NoData      ; проверяем наличие данных
in  AL, DX   ; читаем состояние порта
test AL, 80h ; есть ли данные в порту
jnz @Povtor  ; данных нет
dec DX       ; порт 300h
in  AL, DX   ; читаем порт данных
cmp AL, 0FEh ; успешное подтверждение команды
je  ExitProc ; выходим из процедуры
inc DX       ; порт состояния 301h
@Povtor:
loop @NoData ; делаем еще попытку
```

Аналогичный пример для C++ представлен в листинге 6.25.

Листинг 6.25. Проверка поддержки интерфейса MPU-401 в C++

```
bool IsMPU_401 ()
{
    int iTimeWait = 65535;
    // используем нашу функцию для проверки готовности MIDI
    if ( IsReadMIDI)
      outPort ( 0x301, 0xFF, 1); // записываем команду сброса в порт
    while ( -- iTimeWait > 0)
     {
        // проверяем наличие данных
        if ( IsDataMIDI)
           inPort ( 0x300, &dwResult, 1); // читаем данные
        if ( dwResult == 0xEF) return true;
        // закончилось время ожидания
        if ( iTimeWait < 1) break;
     }
    return false; // MPU-401 отсутствует
}
```

Вторая команда (3Fh) позволяет включить режим UART для обмена данными. Если команда успешно выполнена, интерфейс возвратит значение FEh через порт данных (3x0h). Принцип работы с ней такой же, как и с командой сброса.

Для передачи данных в режиме UART на внешнее устройство MIDI нужно сделать следующее:

1. Проверить готовность устройства MIDI.
2. Если бит 6 равен 0, следует записать данные в порт 3x0h.
3. Если бит 6 равен 1, необходимо перейти к первому пункту.

Для получения данных от внешнего устройства MIDI необходимо выполнить следующие действия:

1. Проверить наличие данных в порту статуса.
2. Если бит 7 равен 0, следует прочитать данные из порта 3x0h.
3. Если бит 7 равен 1, необходимо перейти к первому пункту.

На этом я хотел бы завершить описание работы с составляющими звуковой платы и привести наиболее часто встречающееся расположение базовых адресов (табл. 6.6) и регистров (табл. 6.7).

Таблица 6.6. *Базовые адреса звуковых плат*

Базовый адрес	Диапазон адресов
220h	220h–233h
240h	240h–253h
260h	260h–273h
280h	280h–293h
388h для FM	389h для FM

Таблица 6.7. *Адреса регистров звуковых плат*

Смещение адреса регистра от базового	Описание
0h	Регистр состояния FM (чтение)
0h	Адресный регистр FM (запись)
1h	Регистр данных FM (только запись)
2h	Расширенный регистр состояния FM (чтение)
2h	Расширенный адресный регистр FM (запись)
3h	Расширенный регистр данных FM (только запись)
4h	Адресный регистр микшера (только запись)
5h	Регистр данных микшера (чтение и запись)
6h	Регистр сброса DSP (только запись)
8h	Регистр состояния FM (чтение)
8h	Адресный регистр FM (запись)
9h	Регистр данных FM (только запись)
Ah	Регистр DSP для чтения данных (только чтение)
Ch	Регистр DSP для записи команд и данных (запись)
Ch	Регистр состояния буфера записи (бит 7) DSP (чтение)
Eh	Регистр состояния буфера чтения (бит 7) DSP (только чтение)
10h	Регистр команд или данных для CD-ROM (чтение и запись)
11h	Регистр состояния для CD-ROM (только чтение)
12h	Регистр сброса для CD-ROM (только запись)
13h	Регистр включения CD-ROM (только запись)

6.3. Использование Win32 API

Windows предоставляет программисту возможность работы с микшером звуковой платы и с интерфейсом MIDI. Использование MIDI больше относится к программированию звука, поэтому мы его рассмотрим в следующей главе, а вот управление микшером изучим прямо здесь, поскольку это имеет непосредственное отношение к программированию звуковой платы.

В Windows достаточно удобно работать с микшером, поскольку имеется универсальный программный набор соответствующих функций. Нам не нужно разбираться в портах и особенностях той или иной звуковой платы, а достаточно использовать стандартные функции Win32. Рассмотрим эти функции подробнее:

- `mixerGetDevCaps`. Позволяет получить информацию об установленном микшере;
- `mixerOpen`. Позволяет открыть указанное устройство микшера;
- `mixerClose`. Закрывает указанное устройство микшера;
- `mixerGetID`. Позволяет получить идентификатор для указанного устройства микшера;
- `mixerGetNumDevs`. Определяет количество микшеров, установленных в системе (аппаратных и виртуальных);
- `mixerGetLineInfo`. Позволяет получить информацию об указанной линии (канале) микшера;
- `mixerGetLineControls`. Позволяет получить различные элементы управления (например, ползунки для отображения уровней громкости), расположенные на указанной линии (аудиоканале) микшера;
- `mixerGetControlDetails`. Позволяет получить значения параметров для элементов управления, расположенных на указанной линии (аудиоканале) микшера;
- `mixerSetControlDetails`. Позволяет установить новые значения параметров для элементов управления, расположенных на указанной линии (аудиоканале) микшера.

Для того чтобы разобраться, как все это работает, я приведу пример класса для управления микшером. Назовем наш класс тривиально — `CMixer`. В листинге 6.26 приведен файл CMixer.h. Перед началом работы следует добавить в опции компоновщика ссылку на стандартную библиотеку Winmm.lib. Кроме того, в начало файла поместите ссылку на файл MMsystem.h.

Листинг 6.26. Файл CMixer.h класса `CMixer`

```
// подключаем файл поддержки функций микшера
#include <mmsystem.h>
```

```
// максимальное количество микшеров
#define MAX_MIXER 10
// объявляем наш класс
class CMixer
{
public:
   CMixer ();
   virtual ~CMixer ();
   // поиск первого микшера
   void InitMixer ();
   // функция для установки дескриптора родительского окна
   void SetHWND ( HWND hDlg);
   // функция для получения идентификатора злемента управления
   DWORD GetCtrlID ( int iNum);
   // получение идентификатора открытого микшера
   HMIXER GetMixerID ();
// основные функции работы с микшером
   // управление общим каналом звука
   DWORD GetMinMasterVol (); // получить минимальный уровень громкости
   DWORD GetMaxMasterVol (); // получить максимальный уровень громкости
   DWORD GetTicMasterVol (); // получить значение шага изменения уровня
   DWORD GetMasterValue (); // получить текущее значение громкости
   void SetMasterValue (DWORD dwValue); // установить новое значение
   // управление общим отключением звука
   bool GetMasterMute (); // проверить состояние
   void SetMasterMute ( DWORD dwValue); // установить новое состояние
   // управление фильтром нижних частот (если он есть на звуковой плате)
   DWORD GetMinBass (); // получить минимальное значение
   DWORD GetMaxBass (); // получить максимальное значение
   DWORD GetStepBass (); // получить значение шага изменения уровня
   DWORD GetBassValue (); // получить текущее значение
   void SetBassValue ( DWORD dwValue); // установить новое значение
   // управление фильтром верхних частот (если он есть на плате)
   DWORD GetMinTreble (); // получить минимальное значение
   DWORD GetMaxTreble (); // получить максимальное значение
   DWORD GetStepTreble (); // получить значение шага изменения уровня
   DWORD GetTrebleValue (); // получить текущее значение
   void SetTrebleValue ( DWORD dwValue); // установить новое значение
private:
HWND m_hDlg; // переменная для хранения дескриптора окна
HMIXER Mixer; // дескриптор микшера
// структуры данных для поддержки микшера
MIXERLINE mxl;
MIXERCONTROL mxc;
MIXERLINECONTROLS mxlc;
```

```
// пишем свою структуру для хранения информации о микшере
typedef struct _MixerInfo
{
char Name [MAXPNAMELEN]; // строковое название микшера
unsigned int uID; // идентификатор устройства микшера
WORD manID; // идентификатор производителя микшера
WORD prodID; // идентификатор изделия
DWORD drvVer; // номер версии драйвера
DWORD dwFlags; // дополнительные возможности микшера
DWORD dwLines; // количество доступных линий микшера
} MIXERINFO, *PMIXERINFO;
// структура описания значений параметров для компонентов микшера
typedef struct _Params
{
char name[50]; // имя компонента
DWORD dwID; // идентификатор компонента
DWORD min; // минимальное значение
DWORD max; // максимальное значение
DWORD cur; // текущее значение
DWORD Step; // шаг
} COMPONENTPARAM, *PCOMPONENTPARAM;
// дополнительная структура для описания используемых компонентов
typedef struct _Components
{
    // 0- Master Volume, 1- Mute Master, 2- Bass, 3- Treble
    COMPONENTPARAM comP[4];
} COMP;
// объявляем необходимые структуры
COMP com;
MIXERINFO info;
// и функции
HMIXER OpenMixer ( HWND hwnd, UINT mixerID); // открыть микшер
void CloseMixer (); // закрыть микшер
}; // конец объявления класса Cmixer
```

А теперь посмотрите на файл реализации (CMixer.cpp) класса `CMixer`, представленный в листинге 6.27.

Листинг 6.27. Файл CMixer.cpp класса `CMixer`

```
#include "stdafx.h"
#include "CMixer.h"
```

```
// конструктор класса
CMixer :: CMixer ()
{
 Mixer = NULL;
 m_hDlg = NULL;
 // готовим структуры к использованию
 memset ( &com, 0, sizeof ( COMP));
 memset ( &mxc, 0, sizeof ( MIXERCONTROL));
 memset ( &mxl, 0, sizeof ( MIXERLINE));
 memset ( &mxlc, 0, sizeof ( MIXERLINECONTROLS));
 memset ( &info, 0, sizeof ( MIXERINFO));
 // если микшер в системе не обнаружен, выходим
 if ( !mixerGetNumDevs ()) return;
}
// деструктор класса
CMixer :: ~CMixer ()
{
  // закрываем устройство микшера
  CloseMixer ();
}
// функция для установки дескриптора родительского окна
void SetHWND ( HWND hDlg);
{
   m_hDlg = hDlg
}
// функция поиска и инициализации микшера
void CMixer :: InitMixer ()
{
 if ( m_hDlg == NULL) return;
 // ищем первый завалявшийся микшер
 for ( int i = 0; i < MAX_MIXER; i++)
  {
    if ( !OpenMixer ( m_hDlg, i))
       continue;
    else
       break;
    if ( i > 9) return;
  }
// получаем информацию об общем регуляторе громкости
mxl.cbStruct = sizeof ( MIXERLINE);
mxl.dwComponentType = MIXERLINE_COMPONENTTYPE_DST_SPEAKERS;
mixerGetLineInfo ( ( HMIXEROBJ) Mixer, &mxl, MIXER_OBJECTF_HMIXER |
                        MIXER_GETLINEINFOF_COMPONENTTYPE);
```

```
// получаем параметры для общего регулятора громкости
mxlc.cbStruct = sizeof ( MIXERLINECONTROLS);
mxlc.dwLineID = mxl.dwLineID;
mxlc.dwControlType = MIXERCONTROL_CONTROLTYPE_VOLUME;
mxlc.cControls = 1;
mxlc.cbmxctrl = sizeof ( MIXERCONTROL);
mxlc.pamxctrl = &mxc;
mixerGetLineControls ( ( HMIXEROBJ) Mixer, &mxlc, MIXER_OBJECTF_HMIXER
                                   MIXER_GETLINECONTROLSF_ONEBYTYPE);
// сохраняем полученные значения в нашу структуру
strcpy ( com.comP[0].name, mxl.szName); // имя регулятора громкости
com.comP[0].min = mxc.Bounds.dwMinimum; // минимальное значение
com.comP[0].max = mxc.Bounds.dwMaximum; // максимальное значение
com.comP[0].dwID = mxc.dwControlID; // уникальный идентификатор
com.comP[0].Step = mxc.Metrics.cSteps; // шаг
// получаем информацию о переключателе, управляющего общим звуком
memset ( &mxc, 0, sizeof ( MIXERCONTROL));
memset ( &mxl, 0, sizeof ( MIXERLINE));
memset ( &mxlc, 0, sizeof ( MIXERLINECONTROLS));
mxl.cbStruct = sizeof ( MIXERLINE);
mxl.dwComponentType = MIXERLINE_COMPONENTTYPE_DST_SPEAKERS;

mixerGetLineInfo ( ( HMIXEROBJ) Mixer, &mxl, MIXER_OBJECTF_HMIXER |
                               MIXER_GETLINEINFOF_COMPONENTTYPE);
// получаем параметры для управления переключателем
mxlc.cbStruct = sizeof ( MIXERLINECONTROLS);
mxlc.dwLineID = mxl.dwLineID;
mxlc.dwControlType = MIXERCONTROL_CONTROLTYPE_MUTE;
mxlc.cControls = 1;
mxlc.cbmxctrl = sizeof ( MIXERCONTROL);
mxlc.pamxctrl = &mxc;
mixerGetLineControls ( ( HMIXEROBJ) Mixer, &mxlc, MIXER_OBJECTF_HMIXER
                                   MIXER_GETLINECONTROLSF_ONEBYTYPE);
// сохраняем полученные значения в нашу структуру
strcpy ( com.comP[1].name, mxl.szName); // имя переключателя
com.comP[1].dwID = mxc.dwControlID; // идентификатор переключателя
// получаем информацию о регуляторе нижних частот
memset ( &mxc, 0, sizeof ( MIXERCONTROL));
memset ( &mxl, 0, sizeof ( MIXERLINE));
memset ( &mxlc, 0, sizeof ( MIXERLINECONTROLS));
mxl.cbStruct = sizeof ( MIXERLINE);
mxl.dwComponentType = MIXERLINE_COMPONENTTYPE_DST_SPEAKERS;
mixerGetLineInfo ( ( HMIXEROBJ) Mixer, &mxl, MIXER_OBJECTF_HMIXER |
                               MIXER_GETLINEINFOF_COMPONENTTYPE);
```

```
// получаем параметры для регулятора нижних частот
mxlc.cbStruct = sizeof ( MIXERLINECONTROLS);
mxlc.dwLineID = mxl.dwLineID;
mxlc.dwControlType = MIXERCONTROL_CONTROLTYPE_BASS;
mxlc.cControls = 1;
mxlc.cbmxctrl = sizeof ( MIXERCONTROL);
mxlc.pamxctrl = &mxc;
mixerGetLineControls ( ( HMIXEROBJ) Mixer, &mxlc, MIXER_OBJECTF_HMIXER |
                          MIXER_GETLINECONTROLSF_ONEBYTYPE);
// сохраняем полученные значения в нашу структуру
strcpy (com.comP[2].name, mxl.szName); // имя регулятора нижних частот
com.comP[2].min = mxc.Bounds.dwMinimum; // минимальное значение
com.comP[2].max = mxc.Bounds.dwMaximum; // максимальное значение
com.comP[2].dwID = mxc.dwControlID; // уникальный идентификатор
com.comP[2].Step = mxc.Metrics.cSteps; // шаг
// получаем информацию о регуляторе верхних частот
memset ( &mxc, 0, sizeof ( MIXERCONTROL));
memset ( &mxl, 0, sizeof ( MIXERLINE));
memset ( &mxlc, 0, sizeof ( MIXERLINECONTROLS));
mxl.cbStruct = sizeof ( MIXERLINE);
mxl.dwComponentType = MIXERLINE_COMPONENTTYPE_DST_SPEAKERS;
mixerGetLineInfo ( ( HMIXEROBJ) Mixer, &mxl, MIXER_OBJECTF_HMIXER |
                     MIXER_GETLINEINFOF_COMPONENTTYPE);
// получаем параметры для регулятора верхних частот
mxlc.cbStruct = sizeof ( MIXERLINECONTROLS);
mxlc.dwLineID = mxl.dwLineID;
mxlc.dwControlType = MIXERCONTROL_CONTROLTYPE_TREBLE;
mxlc.cControls = 1;
mxlc.cbmxctrl = sizeof ( MIXERCONTROL);
mxlc.pamxctrl = &mxc;
mixerGetLineControls ( ( HMIXEROBJ) Mixer, &mxlc, MIXER_OBJECTF_HMIXER |
                          MIXER_GETLINECONTROLSF_ONEBYTYPE);
// сохраняем полученные значения в нашу структуру
strcpy (com.comP[3].name, mxl.szName); // имя регулятора верхних частот
com.comP[3].min = mxc.Bounds.dwMinimum; // минимальное значение
com.comP[3].max = mxc.Bounds.dwMaximum; // максимальное значение
com.comP[3].dwID = mxc.dwControlID; // уникальный идентификатор
com.comP[3].Step = mxc.Metrics.cSteps; // шаг
} // конец функции инициализации
// функция для закрытия микшера
void CMixer :: CloseMixer ()
{
    if ( Mixer)
```

```
     {
       // если микшер существует, закрываем его
       mixerClose ( Mixer);
       Mixer = NULL;
      }
}
// функция для получения идентификатора текущего микшера
HMIXER CMixer :: GetMixerID ()
{
    if ( Mixer == NULL)
      return 0;
    return Mixer;
}
// функция для получения идентификатора для указанного компонента
DWORD CMixer :: GetCtrlID ( int iNum)
{
    if ( Mixer == NULL)
      return 0;
    switch ( iNum)
    {
     case 0: // Master Volume
         return com.comP[0].dwID;
     case 1: // Master Mute
         return com.comP[1].dwID;
     case 2: // Bass
         return com.comP[2].dwID;
     case 3: // Treble
         return com.comP[3].dwID;
    }
    return 0;
}
// функция для открытия устройства микшера
HMIXER CMixer :: OpenMixer ( HWND hwnd, UINT mixerID}
{
    UINT uID = 0;
    HMIXER mixerTMP;
    MIXERCAPS mixerCaps;
    // получаем информацию для указанного микшера
    if ( mixerGetDevCaps ( mixerID, &mixerCaps, sizeof ( MIXERCAPS)))
     {
       return ( Mixer); // возвращаем идентификатор предыдущего микшера
     }
    // открываем новое устройство микшера
    if ( mixerOpen ( &mixerTMP, mixerID, (DWORD) hwnd, 0L,
                        CALLBACK_WINDOW))
```

```
     {
       // если не удалось открыть новый микшер, возвращаем предыдущий
       return ( Mixer);
     }
    // если предыдущий микшер был открыт, закрываем его
    if ( Mixer)
     {
       if ( mixerClose ( Mixer))
        {
          // если не удалось закрыть текущий микшер, закрываем новый
          mixerClose ( mixerTMP);
          return ( Mixer);
        }
     }
    // получаем информацию о микшере
    mixerGetID ( ( HMIXEROBJ) Mixer, &uID, MIXER_OBJECTF_HMIXER);
    strcpy ( info.Name, mixerCaps.szPname);
    info.uID = uID;
    info.manID = mixerCaps.wMid;
    info.prodID = mixerCaps.wPid;
    info.drvVer = mixerCaps.vDriverVersion;
    info.dwFlags = mixerCaps.fdwSupport;
    info.dwLines = mixerCaps.cDestinations;
    // сохраняем идентификатор открытого микшера
    Mixer = mixerTMP;
    // выходим из функции
    return ( mixerTMP);
}
// получаем текущее значение для общей громкости
DWORD CMixer :: GetMasterValue ()
{
    MIXERCONTROLDETAILS_UNSIGNED mxcdVolume;
    MIXERCONTROLDETAILS mxcd;
    // получаем текущее значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[0].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_UNSIGNED);
    mxcd.paDetails = &mxcdVolume;
    mixerGetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
                MIXER_OBJECTF_HMIXER | MIXER_GETCONTROLDETAILSF_VALUE);
    com.comP[0].cur = mxcdVolume.dwValue;
```

```
    // возвращаем текущее значение
    return mxcdVolume.dwValue;
}
// функция для установки текущего значения общей громкости
void CMixer :: SetMasterValue ( DWORD dwValue)
{
    MIXERCONTROLDETAILS_UNSIGNED mxcdVolume = { dwValue };
    MIXERCONTROLDETAILS mxcd;
    // устанавливаем новое значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[0].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_UNSIGNED);
    mxcd.paDetails = &mxcdVolume;
    mixerSetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
              MIXER_OBJECTF_HMIXER | MIXER_SETCONTROLDETAILSF_VALUE);
}
// получение минимального значения регулятора общей громкости
DWORD CMixer :: GetMinMasterVol ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[0].min;
}
// получение максимального значения регулятора общей громкости
DWORD CMixer :: GetMaxMasterVol ()
(
    if ( Mixer == NULL)
     return -1;
    return com.comP[0].max;
}
// получить шаг изменения общей громкости
DWORD CMixer :: GetStepMasterVol ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[0].Step;
}
// получение текущего состояния переключателя звука
bool CMixer :: GetMasterMute ()
{
    MIXERCONTROLDETAILS_BOOLEAN mxcdMute;
    MIXERCONTROLDETAILS mxcd;
```

```
    // получаем текущее значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[1].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_BOOLEAN);
    mxcd.paDetails = &mxcdMute;
    mixerGetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
                MIXER_OBJECTF_HMIXER | MIXER_GETCONTROLDETAILSF_VALUE);
    com.comP[1].cur = mxcdMute.fValue;
    // возвращаем текущее значение
    return ( bool) mxcdMute.fValue;
}
// установка состояния переключателя звука
void CMixer :: SetMasterMute ( DWORD dwValue)
{
    MIXERCONTROLDETAILS_BOOLEAN mxcdMute = { dwValue };
    MIXERCONTROLDETAILS mxcd;
    // устанавливаем новое значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[1].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_BOOLEAN);
    mxcd.paDetails = &mxcdMute;
    mixerSetControlDetails (  ( HMIXEROBJ) Mixer, &mxcd,
            MIXER_OBJECTF_HMIXER | MIXER_SETCONTROLDETAILSF_VALUE);
}
// получение текущего значения для регулятора нижних частот
DWORD CMixer :: GetBassValue ()
{
    MIXERCONTROLDETAILS_UNSIGNED mxcdValue;
    MIXERCONTROLDETAILS mxcd;
    // получаем текущее значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[2].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_UNSIGNED);
    mxcd.paDetails = &mxcdValue;
    mixerGetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
            MIXER_OBJECTF_HMIXER | MIXER_GETCONTROLDETAILSF_VALUE);
    com.comP[2].cur = mxcdValue.dwValue;
```

```
    // возвращаем текущее значение
    return mxcdValue.dwValue;
}
// установка нового значения для регулятора нижних частот
void CMixer :: SetBassValue ( DWORD dwValue)
{
    MIXERCONTROLDETAILS_UNSIGNED mxcdVol = { dwValue };
    MIXERCONTROLDETAILS mxcd;
    // устанавливаем новое значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[2].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_UNSIGNED);
    mxcd.paDetails = &mxcdVol;
    mixerSetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
           MIXER_OBJECTF_HMIXER | MIXER_SETCONTROLDETAILSF_VALUE);
}
// получение минимального значения регулятора нижних частот
DWORD CMixer :: GetMinBass ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[2].min;
}
// получение максимального значения регулятора нижних частот
DWORD CMixer :: GetMaxBass ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[2].max;
}
// получить шаг изменения для регулятора нижних частот
DWORD CMixer :: GetStepBass ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[2].Step;
}
// получение текущего значения для регулятора верхних частот
DWORD CMixer :: GetTrebleValue ()
{
    MIXERCONTROLDETAILS_UNSIGNED mxcdValue;
    MIXERCONTROLDETAILS mxcd;
```

```
    // получаем текущее значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[3].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_UNSIGNED);
    mxcd.paDetails = &mxcdValue;
    mixerGetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
                 MIXER_OBJECTF_HMIXER | MIXER_GETCONTROLDETAILSF_VALUE);
    com.comP[3].cur = mxcdValue.dwValue;
    // возвращаем текущее значение
    return mxcdValue.dwValue;
}
// установка нового значения для регулятора верхних частот
void CMixer :: SetTrebleValue ( DWORD dwValue)
{
    MIXERCONTROLDETAILS_UNSIGNED mxcdVol = { dwValue };
    MIXERCONTROLDETAILS mxcd;
    // устанавливаем новое значение
    mxcd.cbStruct = sizeof ( MIXERCONTROLDETAILS);
    mxcd.dwControlID = com.comP[3].dwID;
    mxcd.cChannels = 1;
    mxcd.cMultipleItems = 0;
    mxcd.cbDetails = sizeof ( MIXERCONTROLDETAILS_UNSIGNED);
    mxcd.paDetails = &mxcdVol;
    mixerSetControlDetails ( ( HMIXEROBJ) Mixer, &mxcd,
          MIXER_OBJECTF_HMIXER | MIXER_SETCONTROLDETAILSF_VALUE);
}
// получение минимального значения регулятора верхних частот
DWORD CMixer :: GetMinMasterVol ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[0].min;
}
// получение максимального значения регулятора верхних частот
DWORD CMixer :: GetMaxMasterVol ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[0].max;
}
```

```
// получить шаг изменения для регулятора верхних частот
DWORD CMixer :: GetStepMasterVol ()
{
    if ( Mixer == NULL)
     return -1;
    return com.comP[0].Step;
}
```

Теперь, когда наш класс управления микшером готов, можно добавить его в свою программу. Как вы заметили, мы обработали только четыре управляющих компонента микшера: регулятор общей громкости, переключатель для отключения и включения звука, а также регуляторы низких и высоких частот. Регулировка частот поддерживается не всеми звуковыми платами. Данный класс прекрасно будет работать с семейством звуковых плат от Creative SB Live!. На платах других производителей должна быть поддержка управления низкими и высокими частотами.

Кроме того, разобравшись с работой класса, читатель сможет самостоятельно добавить обработку других стандартных компонентов микшера: регулировка громкости MIDI, аудиодиска, линейного канала. Дополнительную информацию можно получить из документации фирмы Microsoft (на сайте **www.msdn.com**) по программированию звука в Windows, поскольку описать в небольшой главе все возможности просто нереально. Главное — чтобы вы поняли принципы работы с микшером, и тогда сможете самостоятельно разобраться с остальными возможностями.

В завершении темы я приведу примерный код, использующий наш класс `CMixer` (листинг 6.28). Создайте в редакторе ресурсов диалог и добавьте на него три ползунка и один переключатель. Присвойте им следующие идентификаторы: `IDC_VOLUME` (регулятор общей громкости), `IDC_MUTE` (переключатель звука), `IDC_BASS` (регулятор низких частот), `IDC_TREBLE` (регулятор высоких частот).

Листинг 6.28. Использование класса CMixer

```
// Объявляем наш класс
CMixer mixer;
// функция обработки сообщений диалогового окна
BOOL CALLBACK MixerProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam);
// реализация диалоговой функции
BOOL CALLBACK MixerProc (HWND hwndDlg, UINT message, WPARAM wParam,
                         LPARAM lParam)
{
// получаем идентификаторы элементов управления
static HWND hVolume = NULL, hBass = NULL, hTreble = NULL, hMute = NULL;
```

```
hVolume = GetDlgItem ( hwndDlg, IDC_VOLUME);
hMute = GetDlgItem ( hwndDlg, IDC_MUTE);
hBass = GetDlgItem ( hwndDlg, IDC_BASS);
hTreble = GetDlgItem ( hwndDlg, IDC_TREBLE);
// идентификаторы компонентов микшера
static DWORD volID = 0, muteID = 0, bassID = 0, trebleID = 0;
 // обрабатываем сообщения
 switch ( message)
 {
  case WM_INITDIALOG:
   {
    DWORD min = 0, max = 0, cur = 0, Step = 0;
    // устанавливаем идентификатор окна
    mixer.SetHWND ( hwndDlg);
    // инициализируем микшер
    mixer.InitMixer ();
    // получаем идентификаторы компонентов управления микшера
    volID = mixer.GetCtrlID ( 0);
    muteID = mixer.GetCtrlID ( 1);
    bassID = mixer.GetCtrlID ( 2);
    trebleID = mixer.GetCtrlID ( 3);
    // получаем значения параметров микшера
    min = mixer.GetMinMasterVol ();
    max = mixer.GetMaxMasterVol ();
    cur = mixer.GetMasterValue ();
    Step = mixer.GetStepMasterVol ();
    if ( Step == 0) Step = 5000;
    if ( max == 0)
     {
       EnableWindow ( hVolume, false);
     }
    else
     {
       SendMessage ( hVolume, TBM_SETRANGEMIN, ( WPARAM) 0,
                     ( LPARAM) min);
       SendMessage ( hVolume, TBM_SETRANGEMAX, ( WPARAM) 0,
                         ( LPARAM) max);
       SendMessage ( hVolume, TBM_SETTICFREQ, ( WPARAM) max / Step,
                     ( LPARAM) 0);
       SendMessage ( hVolume, TBM_SETPOS, ( WPARAM) 1,
                     ( LPARAM) ( LONG) cur);
     }
    min = 0; max = 0; cur = 0;
    min = mixer.GetMinBass ();
```

```
max = mixer.GetMaxBass ();
cur = mixer.GetBassValue ();
Step = mixer.GetStepBass ();
if ( Step == 0) Step = 5000;
if ( max == 0)
 {
   EnableWindow ( hBass, false);
 )
else
 {
   SendMessage ( hBass, TBM_SETRANGEMIN, ( WPARAM) 0,
                 ( LPARAM) min);
   SendMessage ( hBass, TBM_SETRANGEMAX, ( WPARAM) 0,
                 ( LPARAM) max);
   SendMessage ( hBass, TBM_SETTICFREQ, ( WPARAM) max / Step,
                 ( LPARAM) 0);
   SendMessage ( hBass, TBM_SETPOS, ( WPARAM) 1,
                 ( LPARAM) ( LONG) cur);
 }
min = 0; max = 0; cur = 0;
min = mixer.GetMinTreble ();
max = mixer.GetMaxTreble ();
cur = mixer.GetTrebleValue ();
Step = mixer.GetStepTreble ();
if ( Step == 0) Step = 5000;
if ( max == 0)
 (
   EnableWindow ( hTreble, false);
 }
else
 {
   SendMessage ( hTreble, TBM_SETRANGEMIN, ( WPARAM) 0,
                 ( LPARAM) min);
   SendMessage ( hTreble, TBM_SETRANGEMAX, (WPARAM)0,
                 ( LPARAM) max);
   SendMessage ( hTreble, TBM_SETTICFREQ, ( WPARAM) max / Step,
                 ( LPARAM) 0);
   SendMessage ( hTreble, TBM_SETPOS, (WPARAM)1,
                 ( LPARAM) ( LONG) cur);
 }
if ( muteID == 0)
 {
   EnableWindow ( hMute, false);
 }
```

```
    else
     {
       cur = mixer.GetMasterMute ();
       if ( cur)
        {
          CheckDlgButton ( hwndDlg, IDC_MUTE, true);
          EnableWindow ( hVolume, false);
          EnableWindow ( hBass, false);
          EnableWindow ( hTreble, false);
        }
     }
  }
  return true;
 case WM_COMMAND:
  switch ( LOWORD ( wParam))
   {
    case IDCANCEL:
    case IDOK:
     EndDialog ( hwndDlg, wParam);
     return TRUE;
    // обрабатываем выключатель звука
    case IDC_MUTE:
     if ( HIWORD ( wParam) == BN_CLICKED)
      {
       if ( IsDlgButtonChecked ( hwndDlg, IDC_VOL_MUTE)
                                   == BST_CHECKED)
        {
          mixer.SetMasterMute ( 1};
          EnableWindow { hVolume, false);
          EnableWindow ( hBass, false);
          EnableWindow ( hTreble, false);
        }
       else
        {
          mixer.SetMasterMute ( 0);
          EnableWindow ( hVolume, true);
          EnableWindow ( hBass, true);
          EnableWindow ( hTreble, true);
        }
      }
     break;
   }
```

```
  case WM_HSCROLL:
   {
    // обрабатываем регуляторы
    DWORD pos = 0;
    if ( ( HWND) lParam == hVolume)
     {
       // получаем текущее положение ползунка
       pos = SendMessage ( hVolume, TBM_GETPOS, 0, 0);
       // устанавливаем новое значение
       mixer.SetMasterValue ( pos);
     }
    else if ( ( HWND) lParam == hBass)
     {
       pos = SendMessage ( hBass, TBM_GETPOS, 0, 0);
       mixer.SetBassValue ( pos);
     }
    else if ( ( HWND) lParam == hTreble)
     {
       pos = SendMessage ( hTreble, TBM_GETPOS, 0, 0);
       mixer.SetTrebleValue ( pos);
     }
   }
   break;
 }
 return false;
}
```

Как видите, нет ничего сложного в использовании класса `CMixer`. На этой оптимистической ноте я хотел бы завершить описание программирования звуковой платы и перейти к следующей главе.

ГЛАВА 7

Работа со звуком

В этой главе я хочу познакомить читателя с программированием звука в операционных системах Windows. К сожалению, информации такого рода встречается очень и очень мало. Все так увлеклись базами данных и их составляющими, что совсем забыли о том, что современный компьютер позволяет полностью заменить высококачественный музыкальный центр или звуковую студию, принося гораздо больше пользы, а главное — удовольствия обычным пользователям. Трудно встретить человека, не слушающего музыку. Многие не только слушают, но и сами создают музыкальные произведения. А сколько пользы приносят обучающие компьютерные программы для маленьких детей, развивая не только слух и чувство ритма, но и в игровой форме знакомя с музыкальной азбукой. Что ни говори, а без звука операционные системы Windows потеряли бы половину своей привлекательности и удобства. Без звука трудно себе представить современную игровую или производственную (отслеживающую какие-либо задачи в реальном времени) программу, поэтому я и решил в доступной форме представить читателю различные способы программирования звука в Windows. Мы рассмотрим несколько базовых тем:

1. Создание полноценного плеера аудиодисков средствами MCI (Media Control Interface).
2. Программирование MIDI в Windows.
3. Доступ к файлам в формате MP3.

7.1. Создание плеера аудиодисков

Для того чтобы создать полноценный плеер аудиодисков, нам понадобится всего одна функция MCI. Скажете невозможно? Я бы с вами согласился, если бы не был убежден в обратном. Называется эта функция `mciSendCommand`, и предназначена она для передачи различных предопреде-

ленных сообщений устройству. В качестве устройства могут быть использованы следующие варианты: аудио CD, устройство цифрового воспроизведения звука или видео, MIDI, сканер, видеокассетный магнитофон, проигрыватель видеодисков и стандартное устройство воспроизведения звука. Функция mciSendCommand имеет четыре аргумента:

- IDDevice — идентификатор открываемого устройства. Данный параметр не должен использоваться с сообщением MCI_OPEN.
- uMsg — специальное командное сообщение. Наиболее важные для нас сообщения приведены в табл. 7.1.
- fdwCommand — дополнительные флаги команды. Возможные значения перечислены в табл. 7.2.
- dwParam — указатель на структуру, определяющую параметры команды.

Таблица 7.1. *Командные сообщения*

Имя сообщения	Описание
MCI_OPEN	Позволяет инициализировать любое устройство (файл) MCI
MCI_CLOSE	Закрывает любое устройство (файл) MCI
MCI_GETDEVCAPS	Позволяет получить информацию о любом устройстве MCI
MCI_INFO	Позволяет получить информацию о любом устройстве MCI в виде строки
MCI_PAUSE	Приостанавливает текущее воспроизведение
MCI_RESUME	Возобновляет воспроизведение, приостановленное сообщением MCI_PAUSE
MCI_PLAY	Позволяет начать процесс воспроизведения
MCI_SEEK	Позволяет изменить текущую позицию воспроизведения
MCI_SET	Позволяет настроить доступные параметры устройства
MCI_STATUS	Позволяет получить текущее состояние устройства
MCI_STOP	Останавливает процесс воспроизведения

Таблица 7.2. *Дополнительные флаги*

Имя флага	Описание
MCI_OPEN_TYPE_ID	Определяет, что младшее слово параметра lpstrDeviceType содержит стандартный идентификатор устройства, а старшее — порядковый индекс
MCI_OPEN_TYPE	Тип устройства будет включен в параметр lpstrDeviceType

Таблица 7.2 (окончание)

Имя флага	Описание
MCI_NOTIFY	Позволяет определить обработку уведомляющих сообщений для родительского окна
MCI_OPEN_SHAREABLE	Устройство (файл) будет открыто как общий ресурс
MCI_WAIT	Заставляет устройство возвращать управление программе только после завершения операции
MCI_ALL_DEVICE_ID	Указывает на то, что команда будет передана всем доступным устройствам MCI
MCI_STATUS_MODE	Позволяет установить в параметр dwReturn тип возвращаемого режима работы
MCI_STATUS_ITEM	Позволяет установить тип возвращаемого параметра состояния

Кроме рассмотренной функции, нам понадобятся несколько структур MCI: MCI_OPEN_PARMS, MCI_GENERIC_PARMS, MCI_STATUS_PARMS, MCI_SET_PARMS, MCI_PLAY_PARMS, MCI_SEEK_PARMS и MCI_GETDEVCAPS_PARMS. Каждая из них используется для обработки определенного командного сообщения.

Прежде чем написать код самого плеера, создадим три вспомогательных класса: MCI, MSF и TMSF. Первый класс будет отвечать за инициализацию устройства MCI, а два других помогут осуществить удобное форматирование параметров времени. Для удобства поместим все три класса в общий файл. В листинге 7.1 показан заголовочный файл MCI.h, а в листинге 7.2 — файл реализации MCI.cpp. Не забудьте добавить в опции компоновщика ссылку на библиотеку Winmm.lib.

Листинг 7.1. Файл MCI.h

```
#include <mmsystem.h>
// объявляем класс MSF
class MSF
{
public:
    // конструктор по умолчанию
    MSF () { m_MSF = 0; }
   // дополнительный конструктор с инициализацией
   MSF ( DWORD dwMSF) { m_MSF = dwMSF; }
   // пустой деструктор
   ~MSF () { }
```

```
    // общедоступные функции для форматирования времени
    BYTE GetMin (); // получает составляющую минут
    BYTE GetSec (); // получает составляющую секунд
    BYTE GetFrm (); // получает составляющую фреймов
private:
    // переменная для хранения времени
    DWORD m_MSF;
}; // конец класса MSF
// объявляем класс TMSF
class TMSF
{
public:
// первый конструктор
    TMSF () { m_TMSF = 0; }
    // второй конструктор
    TMSF ( BYTE Track, BYTE Min, BYTE Sec, BYTE Frm)
     {
       // пакуем номер и время трека в DWORD
       m_TMSF = MCI_MAKE_TMSF ( Track, Min, Sec, Frm);
     }
    // пустой деструктор
    ~TMSF () { }
    // общедоступные функции для форматирования времени и номера трека
    BYTE GetTrack (); // возвращает номер трека
    BYTE GetMin (); // получает составляющую минут
    BYTE GetSec (); // получает составляющую секунд
    BYTE GetFrm (); // получает составляющую фреймов
    // определяем оператор для текущего значения
    operator DWORD () { return m_TMSF; }
private:
    // переменная для хранения текущего значения
    DWORD m_TMSF;
}; // конец класса TMSF
// объявляем класс MCI
class MCI
{
public:
    // конструктор по умолчанию и деструктор класса
    MCI ();
    ~MCI ();
    // общедоступные функции
    DWORD OpenMCI ( DWORD dwDevType); // открывает устройство MCI
    DWORD Close (); // закрывает устройство MCI
```

```
   DWORD GetMode (); // получает текущий режим работы устройства MCI
   DWORD GetStatus ( DWORD dwFlag); // получает текущее состояние
   void SetWindow ( HWND hWnd); // устанавливает родительское окно
   MCIERROR GetError (); // получить последнюю ошибку MCI
protected:
   // переменная для хранения идентификатора устройства MCI
   MCIDEVICEID m_MCIID;
   HWND m_hWnd; // дескриптор родительского окна
   DWORD ExecCommand ( unsigned int uCommand, DWORD dwFlag,
                       DWORD dwParam); // выполняет команду MCI
private:
   // переменная для хранения последней ошибки MCI
   MCIERROR m_dwError;
   void FreeMCI (); // закрывает все устройства MCI, освобождая ресурсы
}; // конец класса MCI
```

Листинг 7.2. Файл MCI.cpp

```
#include "stdafx.h"
#include "mci.h"
// реализация класса MSF
BYTE MSF :: GetMin ()
{
   return MCI_MSF_MINUTE ( m_MSF);
}
BYTE MSF :: GetSec ()
{
   return MCI_MSF_SECOND ( m_MSF);
}
BYTE MSF :: GetFrm ()
{
   return MCI_MSF_FRAME ( m_MSF);
}
// окончание реализации класса MSF
// реализация класса TMSF
BYTE TMSF :: GetTrack ()
{
   return MCI_TMSF_TRACK ( m_TMSF);
}
BYTE TMSF :: GetMin ()
{
   return MCI_TMSF_MINUTE ( m_TMSF);
}
```

```
BYTE TMSF :: GetSec ()
{
   return MCI_TMSF_SECOND ( m_TMSF);
}
BYTE TMSF :: GetFrm ()
{
   return MCI_TMSF_FRAME ( m_TMSF);
}
// окончание реализации класса TMSF
// реализация класса MCI
MCI :: MCI () // конструктор класса
{
   m_MCIID = NULL;
   m_hWnd = NULL;
}
// деструктор класса
MCI :: ~MCI ()
{
   if ( m_MCIID)
    {
      FreeMCI ();
      m_MCIID = NULL;
    }
}
// функция для открытия устройства MCI
DWORD MCI :: OpenMCI ( DWORD dwDevType)
{
   MCI_OPEN_PARMS mciOpenParms;
   DWORD dwResult;
   // определяет тип устройства
   mciOpenParms.lpstrDeviceType = (LPCSTR) dwDevType;
   DWORD dwFlags = MCI_OPEN_TYPE_ID | MCI_OPEN_TYPE | MCI_WAIT;
   // открываем устройство
   dwResult = ExecCommand ( MCI_OPEN, dwFlags,
                            ( DWORD) ( LPVOID) &mciOpenParms);
   if ( dwResult == 0)
    {
       // сохраняем идентификатор устройства
       m_MCIID = mciOpenParms.wDeviceID;
    }
   return dwResult;
}
// функция для закрытия устройства MCI
DWORD MCI :: Close ()
```

```
{
   MCI_GENERIC_PARMS mciGenericParms;
   mciGenericParms.dwCallback = (DWORD) m_hWnd;
   return ExecCommand ( MCI_CLOSE, 0, ( DWORD) &mciGenericParms);
}
// возвращает текущий режим работы
DWORD MCI :: GetMode ()
{
   return GetStatus ( MCI_STATUS_MODE);
}
// возвращает состояние для указанного режима
DWORD MCI :: GetStatus ( DWORD dwFlag)
{
   MCI_STATUS_PARMS mciStatusParms;
   mciStatusParms.dwCallback = ( DWORD) m_hWnd;
   mciStatusParms.dwReturn = 0;
   mciStatusParms.dwItem = dwFlag;
   ExecCommand ( MCI_STATUS, MCI_STATUS_ITEM,
               ( DWORD) &mciStatusParms);
   return mciStatusParms.dwReturn;
}
// устанавливает окно, которое будет получать уведомления MCI
void MCI :: SetWindow ( HWND hWnd)
{
   m_hWnd = hWnd;
}
// функция для выполнения указанной команды MCI
DWORD MCI :: ExecCommand ( unsigned int uCommand, DWORD dwFlag,
                           DWORD dwParam)
{
   DWORD dwResult;
   if ( dwResult = mciSendCommand ( m_MCIID, uCommand, dwFlag,
                                    dwParam))
    {
      m_dwError = dwResult;
    }
   return dwResult;
}
// функция возвращает последнюю ошибку MCI
MCIERROR MCI :: GetError () const
{
   return m_dwError;
}
```

```
// функция закрывает все устройства MCI и освобождает ресурсы
void MCI :: FreeMCI ()
{
   mciSendCommand ( MCI_ALL_DEVICE_ID, MCI_CLOSE, MCI_WAIT, NULL);
}
```

Классы `MSF` и `TMSF` помогут нам работать с представлением времени. Для аудиодиска данные можно представить в удобном формате: минута, секунда и фрейм. Дополнительно для операций воспроизведения понадобится еще указать номер трека. Наши классы `MSF` и `TMSF` полностью решают эту задачу. Класс `MCI` является базовым и управляет глобальной работой MCI. С его помощью мы открываем устройство MCI, назначаем дескриптор окна, для которого будут посылаться уведомляющие сообщения MCI, а также закрываем устройство и все используемые системные ресурсы. Этот класс предоставляет универсальную функцию `ExecCommand`, с помощью которой выполняется передача всех управляющих команд устройству.

Теперь у нас все готово для создания класса аудиоплеера. Назовем его `CDAudio` и сделаем производным от `MCI`. Добавим необходимые функции для управления плеером. Файл CDAudio.h показан в листинге 7.3.

Листинг 7.3. Файл CDAudio.h

```
#include "mci.h"
// объявляем наш класс
class CDAudio : public MCI
{
public:
   // определяем конструктор по умолчанию
   CDAudio ();
   // определяем пустой деструктор
   ~ CDAudio ();
   // определяем общедоступные функции класса
   DWORD Open (); // открываем устройство CD Audio
   // функции для управления дверцей CD-ROM
   void OpenTray (); // открыть
   void CloseTray (); // закрыть
   // функции для управления воспроизведением
   void PlayCD ( int TrackStart, int MinStart, int SecStart,
   int TrackEnd, int MinEnd, int SecEnd); // играть от и до
   void PlayStartEndCD ( DWORD dwStart, DWORD dwEnd);
   void PauseCD (); // приостановить воспроизведение
   void ResumeCD (); // продолжить воспроизведение
```

```
  // перемещение по диску
  void SeekToStart (); // в начало диска
  void SeekToEnd (); // в конец диска
  // проверка состояния
  bool IsDisk (); // есть ли диск
  bool IsReady (); // готово ли устройство
  // управление форматом представления данных
  void GetFormat (); // получить текущий формат
  void SetFormat ( DWORD dwFormat); // установить новый формат
  // получение различной информации
  DWORD GetTrackType ( DWORD dwTrack); // тип трека
  DWORD GetTrackLen ( DWORD dwTrack); // длина трека
  DWORD GetTrackPosition ( DWORD dwTrack); // позиция трека
  DWORD GetCountTracks (); // количество треков на диске
  DWORD GetTotalLenCD (); // полный размер диска
  DWORD GetCurPosition (); // текущая позиция
  DWORD GetCurTrack (); // текущий трек
private:
  // сервисные функции
  DWORD _getTrackInfo ( DWORD dwTrack, DWORD dwFlag); // информация
  DWORD _seek ( DWORD dwPos, DWORD dwFlag); // перемещение по диску
}; // окончание класса CDAudio
```

Как вы видите, в классе определены наиболее важные функции управления нашим устройством. На их основе можно реализовать практически любые возможности для аудиоплеера. Файл реализации (CDAudio.cpp) класса `CDAudio` представлен в листинге 7.4.

Листинг 7.4. Файл CDAudio.cpp

```
#include "stdafx.h"
#include "CDAudio.h"
// конструктор класса
CDAudio :: CDAudio ();
{
}
// сервисные функции
DWORD CDAudio :: _getTrackInfo ( DWORD dwTrack, DWORD dwFlag)
{
  MCI_STATUS_PARMS mciStatus;
  mciStatus.dwTrack = dwTrack;
  mciStatus.dwItem = dwFlag;
  mciStatus.dwReturn = 0;
```

```
   mciStatus.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_STATUS, MCI_TRACK | MCI_STATUS_ITEM,
                ( DWORD) &mciStatus);
   return mciStatus.dwReturn;
}
DWORD CDAudio :: _seek ( DWORD dwPos, DWORD dwFlag)
{
   MCI_SEEK_PARMS mciSeek;
   mciSeek.dwTo = dwPos;
   dwFlag |= MCI_NOTIFY;
   mciSeek.dwCallback = ( DWORD) m_hWnd;
   return ExecCommand ( MCI_SEEK, dwFlag, ( DWORD) &mciSeek);
}
// общие функции управления и поддержки
DWORD CDAudio :: Open ()
{
   // открываем устройство CD аудио
   return OpenMCI ( MCI_DEVTYPE_CD_AUDIO);
}
void CDAudio :: OpenTray ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_SET, MCI_SET_DOOR_OPEN, ( DWORD) &mciSet);
}
void CDAudio :: CloseTray ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_SET, MCI_SET_DOOR_CLOSED, ( DWORD) &mciSet);
}
void CDAudio :: PlayCD ( int TrackStart, int MinStart, int SecStart,
                         int TrackEnd, int MinEnd, int SecEnd)
{
   MCI_PLAY_PARMS mciPlay;
   mciPlay.dwFrom = MCI_MAKE_TMSF ( TrackStart, MinStart, SecStart, 0);
   mciPlay.dwTo = MCI_MAKE_TMSF ( TrackEnd, MinEnd, SecEnd, 0);
   DWORD dwFlag = MCI_NOTIFY | MCI_FROM | MCI_TO;
   mciPlay.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_PLAY, dwFlag, ( DWORD) ( LPVOID) &mciPlay);
}
void CDAudio :: PlayStartEndCD ( DWORD dwStart, DWORD dwEnd)
{
   MCI_PLAY_PARMS mciPlay;
   mciPlay.dwFrom = dwStart;
```

```
   mciPlay.dwTo = dwEnd;
   DWORD dwFlag = MCI_NOTIFY | MCI_FROM | MCI_TO;
   mciPlayParms.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_PLAY, dwFlag, ( DWORD) ( LPVOID) &mciPlay);
}
void CDAudio :: PauseCD ()
(
   MCI_GENERIC_PARMS mciGeneric;
   mciGeneric.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_PAUSE, 0, ( DWORD) &mciGeneric);
}
void CDAudio :: ResumeCD ()
{
MCI_GENERIC_PARMS mciGeneric;
   mciGeneric.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_RESUME, 0, ( DWORD) &mciGeneric);
}
void CDAudio :: SeekToStart ()
{
   return _seek( 0, MCI_SEEK_TO_START);
}
void CDAudio :: SeekToEnd ()
{
   return _seek( 0, MCI_SEEK_TO_END);
}
bool CDAudio :: IsDisk ()
{
   return GetStatus ( MCI_STATUS_MEDIA_PRESENT);
}
bool CDAudio :: IsReady ()
{
   return GetStatus ( MCI_STATUS_READY);
}
void CDAudio :: GetFormat ()
{
   return GetStatus ( MCI_STATUS_TIME_FORMAT);
}
void CDAudio :: SetFormat ( DWORD dwFormat)
{
   MCI_SET_PARMS mciSet;
   mciSet.dwTimeFormat = dwFormat;
   ExecCommand ( MCI_SET, MCI_SET_TIME_FORMAT,
               ( DWORD) ( LPVOID) &mciSet);
}
```

```
DWORD CDAudio.:: GetTrackType ( DWORD dwTrack)
{
   return _getTrackInfo ( dwTrack, MCI_CDA_STATUS_TYPE_TRACK);
}
DWORD CDAudio :: GetTrackLen ( DWORD dwTrack)
{
   return _getTrackInfo ( dwTrack, MCI_STATUS_LENGTH);
}
DWORD CDAudio :: GetTrackPosition ( DWORD dwTrack)
{
   return _getTrackInfo ( dwTrack, MCI_STATUS_POSITION);
}
DWORD CDAudio :: GetCountTracks ()
{
   return GetStatus ( MCI_STATUS_NUMBER_OF_TRACKS);
}
DWORD CDAudio :: GetTotalLenCD ()
{
   return GetStatus ( MCI_STATUS_LENGTH);
}
DWORD CDAudio :: GetCurPosition ()
{
   return GetStatus ( MCI_STATUS_POSITION);
}
DWORD CDAudio :: GetCurTrack ()
{
   return GetStatus ( MCI_STATUS_CURRENT_TRACK);
}
```

На этом наш класс CDAudio можно считать полностью законченным. Вам осталось только создать красивый интерфейс и подключить к нему CDAudio. Я не буду приводить полный код программы воспроизведения аудиодисков, а покажу только наиболее интересные моменты в создании плеера. Перед началом работы необходимо инициализировать класс CDAudio, как показано в листинге 7.5. Сразу хочу заметить, что в примерах подразумевается поддержка библиотеки MFC.

Листинг 7.5. Инициализация класса CDAudio и получение информации о треках

```
// предположим, что класс плеера объявлен как CDAudio cd;
// функция инициализации класса CDAudio
bool CMyPlayer :: Init ()
```

```
{
  if ( cd.Open ())
   return true;
  else
  {
   cd.SetWindow ( hMainWnd); // указываем главное окно программы
   if ( !cd.IsReady ())
   {
     // устройство не готово
   }
   // устанавливаем формат представления данных
   cd.SetFormat ( MCI_FORMAT_TMSF);
   // загружаем информацию о треках в список List View
   LoadTracks ();
  }
  return false;
}
// функция для загрузки треков в окно программы
// предположим, что наш List View назван m_TrackList
void CMyPlayer :: LoadTracks ()
{
  CString info;
  MSF msf;
  char Text[30];
  DWORD dwTracks = 0;
  DWORD dwSize = 0;
  // предварительно очищаем список
  m_TrackList.DeleteAllItems ();
  // определяем общее число треков на диске
  dwTracks = cd.GetCountTracks ();
  for ( UINT i = 1; i <= dwTracks; i++)
  {
     // получаем длину первого трека
     msf = cd.GetTrackLength ( i);
     // форматируем строку в удобную для нас форму
     info.Format ( "%02u : %02u", msf.GetMin (), msf.GetSec ());
     // создаем имя для трека
     sprintf ( Text, "  Трек %u", i);
     // выводим информацию в List View
     m_TrackList.SetItemText ( i - 1, 0, Text);
     m_TrackList.SetItemText ( i - 1, 1, info);
     // считаем размер трека в мегабайтах
     dwSize = ( ( msf.GetMin () * 60) + msf.GetSec ());
      dwSize *= 75; // число секторов в одной секунде
```

```
        dwSize *= 2352; // размер в байтах одного сектора
        sprintf ( Text, "%ld", dwSize / 1048576);
        m_TrackList.SetItemText ( i - 1, 2, Text);
    }
}
```

А чтобы открыть лоток устройства, можно написать функцию, приведенную в листинге 7.6.

Листинг 7.6. Пример функции для открывания лотка CD-ROM

```
void CMyPlayer :: Door ( bool bOpen)
{
   if ( bOpen) // открыть
    cd.OpenTray ();
   else
    cd.CloseTray ();
}
```

Реализовать функцию паузы можно так, как это показано в листинге 7.7. Сделана она таким образом, что позволит включить паузу при первом вызове и продолжить воспроизведение при втором вызове.

Листинг 7.7. Пример функции для приостановки воспроизведения

```
void CMyPlayer :: Pause ()
{
   if ( cd.GetMode () == MCI_MODE_PLAY) // идет воспроизведение
   {
     // устанавливаем паузу и останавливаем таймер воспроизведения
     cd.PauseCD ();
     KillTimer ( MY_TIMER);
   }
   else
   {
      // продолжаем воспроизведение и запускаем таймер
      cd.ResumeCD ();
      SetTimer ( MY_TIMER, 1000, NULL);
   }
}
```

Использовать таймер в программе необходимо, если вы хотите отслеживать время воспроизведения и отображать это в своей программе. Для воспроизведения трека можно применить функцию из листинга 7.8.

Листинг 7.8. Пример функции воспроизведения

```
void CMyPlayer :: Play ( BYTE Track)
{
  TMSF m_Start;
  TMSF m_End;
  // определяем длину трека
  MSF msf = cd.GetTrackLen ( Track);
  m_Start = TMSF ( Track, 0, 0, 0); // начальная позиция
  m_End = TMSF ( Track, msf.GetMin (), msf.GetSec (),
              msf.GetFrm ()); // конечная позиция
  cd.PlayStartEndCD ( m_Start, m_End); // воспроизведение
  SetTimer ( MY_TIMER, 1000, NULL); // включаем таймер
}
```

При завершении работы программы следует правильно закрыть все ресурсы. Как это делается, показано в листинге 7.9.

Листинг 7.9. Завершение работы программы

```
void CMyPlayer :: OnCancel ()
{
  if ( MessageBox ( "Выйти из программы ?"), "Завершение работы",
       MB_ICONQUESTION | MB_OKCANCEL | MB_DEFBUTTON1) == IDOK)
 {
  if (cd.IsReady ())
  {
    cd.StopCD (); // останавливаем, если нужно воспроизведение
    cd.Close (); // закрываем устройство MCI
    KillTimer ( MY_TIMER); // выключаем таймер
  }
  CDialog :: OnCancel ();
 }
}
```

И еще я хотел бы рассказать, как организовать быстрое перемещение по музыкальному треку во время воспроизведения. Предположим, что вы добавили ползунок в редакторе ресурсов, назвали его `IDC_POSITION` и определили для него переменную `m_Position` (класс `CSliderCtrl` из библиотеки MFC). После этого добавьте обработку сообщения `WM_HSCROLL` и заполните его так, как это сделано в листинге 7.10. Переменную `dwTrackLenght` следует определить заранее в классе так, чтобы она определяла полный размер текущего трека, а переменная `m_CurTrack` должна хранить номер текущего трека.

Листинг 7.10. Обработка сообщения WM_SCROLL

```
void CMyPlayer :: OnHScroll ( UINT nSBCode, UINT nPos,
                              CScrollBar* pScrollBar)
{
   if ( pScrollBar->GetDlgCtrlID () == IDC_POSITION)
   {
     // получаем текущую позицию
     DWORD dwCurPosition = m_Position.GetPos ();
     // вычисляем новую позицию
     DWORD dwTimePosition = dwTrackLenght / 1000;
     cd.PlayTrackCD ( m_CurTrack, dwCurPosition / 60, dwCurPosition % 60,
                 m_CurTrack, dwTimePosition / 60, dwTimePosition % 60);
   }
   Cdialog :: OnHScroll ( nSBCode, nPos, pScrollBar);
}
```

На этом можно завершить описание работы с MCI для программирования устройства компакт-дисков.

7.2. Программирование MIDI

Интерфейс MIDI представляет собой цифровой протокол, основанный на передаче так называемых событий MIDI (MIDI event). С помощью этих событий можно управлять различными устройствами (например, синтезатором), подключенными к игровому порту звуковой карты. Еще раз напомню, что для корректной работы внешнего устройства необходим дополнительный переходник. По протоколу интерфейса MIDI можно подключать последовательно несколько устройств. Для дополнительной поддержки MIDI в 1988 году был разработан специальный файловый формат (Standard MIDI File Format), который четко оговаривал хранение событий MIDI и синхронизирующей информации в одном файле. Поскольку файл содержал не саму музыку, а только управляющие команды для звукового процессора, размер файла получался очень небольшим. Однако в этом заключался и основной недостаток данного формата: на разных по возможностям звуковых платах файл звучал неодинаково. Для того чтобы хоть немного исправить эту проблему, был принят дополнительный стандарт General MIDI, определяющий звучание для 175 инструментов. Теперь любой файл звучит одинаково (по составу инструментов) на любой звуковой плате, однако качество звука все равно может сильно отличаться. Не буду вдаваться в подробности данной проблемы, поскольку книга все-таки рассматривает программирование, а не создание музыки.

Исходя из вышеизложенного, хочу представить вашему вниманию простой и удобный способ воспроизведения MIDI-файлов. Если вы планируете добавить в программу только возможность прослушивания таких файлов, этот вариант будет наиболее оптимальным.

Итак, создадим новый класс и назовем его `MIDI`. Заполните определение класса, согласно листингу 7.11.

Листинг 7.11. Файл MIDI.h класса `MIDI`

```
#include "Mmsystem.h"
// объявление класса MIDI
class MIDI
{
public:
    // конструктор по умолчанию
    MIDI ();
    // пустой деструктор
    ~MIDI () { }
   // общие функции
   void Open ( char* szFileMidi); // загружает файл MIDI, открывает MCI
   void Play (); // воспроизводит MIDI-файл
   void Stop (); // останавливает воспроизведение
   void Close (); // закрывает устройство MCI
private:
// буфер для хранения текстовой строки
char szBuffer[350];
// указатель на загруженный файл
bool bLoad;
}; // окончание класса MIDI
```

Мы определили три функции: для открытия файла, воспроизведения и остановки воспроизведения. Теперь напишем файл реализации класса `MIDI` так, как показано в листинге 7.12.

Листинг 7.12. Файл MIDI.cpp класса `MIDI`

```
#include "stdafx.h"
#include "MIDI.h"
// конструктор по умолчанию
MIDI :: MIDI ()
{
   bLoad = false;
}
```

```
// функция Open
MIDI :: Open ( char* szFileMidi)
{
   // сохраняем в классе указанный файл
   if ( szFileMidi)
   {
     // форматируем командную строку
     sprintf ( szBuffer, "open \"%s\" type sequencer alias MIDIPLAYER",
             szFileMidi);
     // и передаем ее функции MCI
     mciSendString ( szBuffer, 0, 0, 0);
     bLoad = true;
   }
}
// функция Play
MIDI :: Play ()
{
   if ( !bLoad) return;
   // воспроизводим файл MIDI
   mciSendString ( "play MIDIPLAYER from 0", 0, 0, 0);
}
// функция Stop
MIDI :: Stop ()
{
   if ( !bLoad) return;
   // останавливаем воспроизведение файла MIDI
   mciSendString ( "stop MIDIPLAYER", 0, 0, 0);
}
// функция Close
MIDI :: Close ()
{
   bLoad = false;
   // закрываем устройство MCI
   mciSendString ( "close all", 0, 0, 0);
}
```

Вот и все. Класс полностью готов к работе. С его помощью вы можете прослушать следующие типы файлов: mid, midi и rmi. В качестве основной функции выступает стандартная функция MCI `mciSendString`. Она выполняет те же задачи, что и рассмотренная ранее `mciSendCommand`. Главным отличием является представление аргументов: все управляющие команды записываются в виде текстовой строки.

Написать плеер для воспроизведения MIDI-файлов можно и на базе рассмотренного ранее класса `MCI`. Создадим еще один класс `CMidi`, производ-

ный от класса MCI, и определим набор функций, как показано в листинге 7.13.

Листинг 7.13. Файл CMidi.h

```
#include "mci.h"
// объявляем класс
class CMidi : public MCI
{
public:
// конструктор по умолчанию и деструктор класса
   CMidi ();
   ~ CMidi () { }
   // общедоступные функции управления воспроизведением
   void OpenMIDI ( LPCSTR lpszMidiFile); // открывает устройство MIDI
   void PlayMIDI (); // воспроизведение файла MIDI
   void PauseMIDI (); // пауза
   void ResumeMIDI (); // продолжение воспроизведения после паузы
   void StopMIDI (); // остановка воспроизведения
   void SeekToStart (); // перейти в начало файла
   void SeekToEnd (); // перейти в конец файла
   // функции для получения информации
   bool IsReady (); // готово ли устройство
   DWORD GetMidiLen (); // длина композиции MIDI
   DWORD GetCurPosition (); // текущая позиция воспроизведения
   // дополнительные функции
   DWORD GetFormat (); // получить текущий формат времени
   void SetFormatMs (); // миллисекунды
   void SetFormatSMPTE_24 (); // 24 кадра
   void SetFormatSMPTE_25 (); // 25 кадров
   void SetFormatSMPTE_30 (); // 30 кадров
   void SetFormatSMPTE_30DROP (); // 30 кадров
   DWORD GetMidiTempo (); // получить текущий темп
   void SetMidiTempo ( DWORD dwTempo); // установить новый темп
   void SetMidiCurPort (); // выбрать MIDI-порт по умолчанию
   void SetMidiMapperPort (); // выбрать MIDI mapper порт
   void SaveMidi ( LPCSTR lpszMidiFile); // сохранить в файл
private:
   void _seek ( DWORD dwEnd, DWORD dwFlag); // установка позиции
}; // окончание класса CMidi
```

Теперь напишем реализацию функций для класса CMidi (листинг 7.14).

Листинг 7.14. Файл CMidi.cpp

```
#include "stdafx.h"
#include "CMidi.h"
// реализация класса CMidi
CMidi :: CMidi () // конструктор класса
{
}
void CMidi :: OpenMIDI ( LPCSTR lpszMidiFile)
{
   MCI_OPEN_PARMS mciOpen;
   mciOpen.lpstrElementName = lpszMidiFile;
   mciOpen.lpstrDeviceType = ( LPCSTR) MCI_DEVTYPE_SEQUENCER;
   DWORD dwFlag = MCI_WAIT | MCI_OPEN_ELEMENT |
                  MCI_OPEN_TYPE|MCI_OPEN_TYPE_ID;
   ExecCommand ( MCI_OPEN, dwFlag, ( DWORD) ( LPVOID) &mciOpen);
   // сохраняем идентификатор открытого устройства
   m_MCIID = mciOpen.wDeviceID;
}
void CMidi :: PlayMIDI ()
{
   MCI_PLAY_PARMS mciPlay;
   mciPlay.dwCallback = ( DWORD) m_hWnd;
   DWORD dwFlag |= MCI_NOTIFY;
   ExecCommand ( MCI_PLAY, dwFlag, ( DWORD) ( LPVOID) &mciPlay);
}
void CMidi :: PauseMIDI ()
{
   MCI_GENERIC_PARMS mciGeneric;
   mciGeneric.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_PAUSE, 0, ( DWORD) &mciGeneric);
}
void CMidi :: ResumeMIDI ()
{
   MCI_GENERIC_PARMS mciGeneric;
   mciGeneric.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_RESUME, 0, ( DWORD) &mciGeneric);
}
void CMidi :: StopMIDI ()
{
   MCI_GENERIC_PARMS mciGeneric;
   mciGeneric.dwCallback = ( DWORD) m_hWnd;
   ExecCommand ( MCI_STOP, 0, ( DWORD) &mciGeneric);
}
```

```
void CMidi :: SeekToStart ()
{
   return _seek ( 0, MCI_SEEK_TO_START);
}
void CMidi :: SeekToEnd ()
{
   return _seek ( 0, MCI_SEEK_TO_END);
}
void CMidi :: _seek ( DWORD dwEnd, DWORD dwFlag)
{
   MCI_SEEK_PARMS mciSeek;
   mciSeek.dwCallback = ( DWORD) m_hWnd;
   mciSeek.dwTo = dwEnd;
   dwFlag |= MCI_NOTIFY;
   ExecCommand ( MCI_SEEK, dwFlag, ( DWORD) &mciSeek);
}
bool CMidi :: IsReady ()
{
   return GetStatus ( MCI_STATUS_READY);
}
DWORD CMidi :: GetMidiLen ()
{
   return GetStatus ( MCI_STATUS_LENGTH);
}
DWORD CMidi :: GetCurPosition ()
{
   return GetStatus ( MCI_STATUS_POSITION);
}
DWORD CMidi :: GetFormat ()
{
   return GetStatus ( MCI_STATUS_TIME_FORMAT);
}
void CMidi :: SetFormatMs ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwTimeFormat = MCI_FORMAT_MILLISECONDS;
   ExecCommand ( MCI_SET, MCI_SET_TIME_FORMAT,
               ( DWORD) ( LPVOID) &mciSet);
}
void CMidi :: SetFormatSMPTE_24 ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwTimeFormat = MCI_FORMAT_SMPTE_24;
```

```
   ExecCommand ( MCI_SET, MCI_SET_TIME_FORMAT,
                 ( DWORD) ( LPVOID) &mciSet);
}
void CMidi :: SetFormatSMPTE_25 ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwTimeFormat = MCI_FORMAT_SMPTE_25;
   ExecCommand ( MCI_SET, MCI_SET_TIME_FORMAT,
                 ( DWORD) ( LPVOID) &mciSet);
}
void CMidi :: SetFormatSMPTE_30 ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwTimeFormat = MCI_FORMAT_SMPTE_30;
   ExecCommand ( MCI_SET, MCI_SET_TIME_FORMAT,
                 ( DWORD) ( LPVOID) &mciSet);
}
void CMidi :: SetFormatSMPTE_30DROP ()
{
   MCI_SET_PARMS mciSet;
   mciSet.dwTimeFormat = MCI_FORMAT_SMPTE_30DROP;
   ExecCommand ( MCI_SET, MCI_SET_TIME_FORMAT,
                 ( DWORD) ( LPVOID) &mciSet);
}
DWORD CMidi :: GetMidiTempo ()
{
   return GetStatus ( MCI_SEQ_STATUS_TEMPO);
}
void CMidi :: SetMidiTempo ( DWORD dwTempo)
{
   return GetStatus ( MCI_SEQ_STATUS_TEMPO);
}
void CMidi :: SetMidiCurPort ()
{
   MCI_SEQ_SET_PARMS mciSeqSet;
   mciSeqSet.dwPort = MCI_SEQ_NONE;
   ExecCommand ( MCI_SET, MCI_SEQ_SET_PORT,
                 ( DWORD) ( LPVOID) &mciSeqSet);
}
void CMidi :: SetMidiMapperPort ()
{
   MCI_SEQ_SET_PARMS mciSeqSet;
   mciSeqSet.dwPort = MIDI_MAPPER;
```

```
   ExecCommand ( MCI_SET, MCI_SEQ_SET_PORT,
               ( DWORD) ( LPVOID) &mciSeqSet);
}
void CMidi :: SaveMidi ( LPCSTR lpszMidiFile)
{
   MCI_SAVE_PARMS mciSave;
   mciSave.lpfilename = lpszMidiFile;
   ExecCommand ( MCI_SAVE, MCI_SAVE_FILE, ( DWORD) &mciSave);
}
```

Вот у нас и получился полноценный класс для воспроизведения MIDI-файлов. На этом работу с MIDI будем считать завершенной.

7.3. Доступ к файлам в формате MP3

Говорить о данном формате не имеет смысла, поскольку с ним знаком практически каждый. Как реализовать поддержку кодирования и декодирования файлов MP3 в программе, рассказывать не буду, поскольку эту информацию можно найти в Интернете. Мы поговорим о том, как воспроизвести файл MP3, а также как модифицировать информационный заголовок в таком файле.

Для воспроизведения MP3 воспользуемся уже знакомым нам интерфейсом MCI. Напишем новый класс плеера и назовем его `CMp3`. В опции компоновщика следует добавить ссылку на библиотеку Vfw32.lib. Заголовочный файл нашего класса представлен в листинге 7.15.

Листинг 7.15. Файл CMp3.h

```
#include <vfw.h>
// объявление класса CMp3
class CMp3
{
public:
   CMp3 (); // конструктор по умолчанию
   ~CMp3 (); // деструктор
   // общие функции управления
   void Open (); // открыть файл MP3
   void Play (); // воспроизведение файла
   void Pause (); // пауза
   void Stop (); // стоп
   // установить параметры родительского окна
   void SetWinParam ( HWND hWnd, HINSTANCE hInst);
```

```
private:
    HWND m_hWnd; // дескриптор родительского окна
    HINSTANCE m_hInst; // дескриптор вызывающего приложения
    HWND m_MP3; // дескриптор устройства
    bool m_bPause; // статус воспроизведения
};
```

Теперь напишем реализацию класса, согласно листингу 7.16.

Листинг 7.16. Файл CMp3.cpp

```
#include "stdafx.h"
#include "CMp3.h"
#include "vfw.h"
// реализация класса CMp3
CMp3 :: CMp3 ()
{
   m_hWnd = NULL;
   m_MP3 = NULL;
   m_hInst = NULL;
   m_bPause = false;
}
CMp3 :: ~CMp3 ()
{
   // освобождаем ресурсы
   if ( m_MP3) MCIWndDestroy ( m_MP3);
}
void CMp3 :: SetWinParam ( HWND hWnd, HINSTANCE hInst)
{
  if ( hWnd && hInst)
  {
    m_hWnd = hWnd;
    m_hInst = hInst;
  }
}
void CMp3 :: Open ()
{
   if ( ( !m_hWnd) && ( !m_hInst)) return;
   CFileDialog file ( TRUE, NULL, NULL, OFN_HIDEREADONLY,
                     "MP3 Files (*.mp3)|*.mp3||");
   if ( file.DoModal () == IDOK)
   {
```

```
      // открываем устройство MCI
      m_MP3 = MCIWndCreate ( m_hWnd, m_hInst, WS_CHILD
              | MCIWNDF_NOMENU | MCIWNDF_NOPLAYBAR, file.m_ofn.lpstrFile);
   }
}
void CMp3 :: Play ()
{
   if( !m_MP3) return;
   MCIWndHome ( m_MP3); // исходное положение файла
   MCIWndPlay ( m_MP3); // начинаем воспроизведение
}
void CMp3 :: Pause ()
{
   if ( !m_MP3) return;
   if ( m_bPause)
   {
     MCIWndResume ( m_MP3);
     m_bPause = false;
   }
   else
   {
     MCIWndPause ( m_MP3);
     m_bPause = true;
   }
}
void CMp3 :: Stop ()
{
   if( !m_MP3) return;
   MCIWndStop ( m_MP3);
   m_bPause = false;
}
```

Вот у нас и получился простейший класс для воспроизведения файлов в формате MP3. Все достаточно наглядно и не требует дополнительных комментариев. Хочу сказать только два слова о назначении функции `SetWinParam`. С помощью данной функции мы устанавливаем для нашего класса дескрипторы родительского окна и приложения, поскольку этого требует интерфейс MCI. После объявления класса `CMp3` следует сразу вызвать функцию `SetWinParam`, а только потом использовать управляющие функции.

А теперь рассмотрим, как можно прочитать и модифицировать информационный заголовок файла MP3. Для этого потребуется написать класс

CMp3Info, но прежде познакомимся с заголовком файла MP3. Формат заголовка приведен в табл. 7.3.

Таблица 7.3. *Заголовок файла MP3*

Байт	Биты	Описание
0	8	Биты синхронизации установлены в 1
1	3	Биты синхронизации установлены в 1
	2	Номер версии
	2	Индекс уровня (1 — Layer 3, 2 — Layer 2, 3 — Layer 1)
	1	Защита контрольной суммой (CRC)
2	4	Скорость передачи (битов в секунду)
	2	Частота дискретизации
	1	При установке в 1 используется добавочный слот
	1	Частный
3	2	Режим (0 — стерео, 1 — объединенное стерео, 2 — два канала, 3 — один канал)
	2	Расширенный режим
	1	Авторское право (1 — есть)
	1	Оригинал (1 — оригинальная композиция)
	2	Предыскажение

Кроме перечисленного в таблице формата, файл может содержать дополнительные сведения (128 байт): полное имя файла, данные об исполнителе, название альбома, стиль композиции и дату создания.

Рассмотрим первый файл класса (CMp3Info.h), представленный в листинге 7.17.

Листинг 7.17. Файл CMp3Info.h

```
// сначала определим список музыкальных стилей
static char* Genres [] = { "Blues", "Classic Rock", "Country", "Dance",
"Disco", "Funk", "Grunge", "Hip-Hop", "Jazz", "Metal", "New Age",
"Oldies", "Other", "Pop", "R&B", "Rap", "Reggae", "Rock", "Techno",
"Industrial", "Alternative", "Ska", "Death Metal", "Pranks",
"Soundtrack", "Euro-Techno", "Ambient", "Trip Hop", "Vocal", "Jazz+Funk",
"Fusion", "Trance", "Classical", "Instrumental", "Acid", "House", "Game",
"Sound Clip", "Gospel", "Noise", "Alternative Rock", "Bass", "Soul",
```

```
"Punk", "Space", "Meditative", "Instrumental Pop", "Instrumental Rock",
"Ethnic", "Gothic", "Darkwave", "Techno-Industrial", "Electronic",
"Pop-Folk", "Eurodance", "Dream", "Southern Rock", "Comedy", "Cult",
"Gangsta Rap", "Top 40", "Christian Rap", "Pop/Punk", "Jungle",
"Native American", "Cabaret", "New Wave", "Psychedelic", "Rave",
"Showtunes", "Trailer", "Lo-Fi", "Tribal", "Acid Punk", "Acid Jazz",
"Polka", "Retro", "Musical", "Rock & Roll", "Hard Rock", "Folk",
"Folk/Rock", "National Folk", "Swing", "Fast-Fusion", "Bebob", "Latin",
"Revival", "Celtic", "Blue Grass", " Avantgarde", "Gothic Rock",
"Progressive Rock", "Psychedelic Rock", "Symphonic Rock", "Slow Rock",
"Big Band", "Chorus", "Easy Listening", "Acoustic", "Humour", "Speech",
"Chanson", "Opera", "Chamber Music", "Sonata", "Symphony", "Booty Bass",
"Primus", "Porn Groove", "Satire", "Slow Jam", "Club", "Tango", "Samba",
"Folklore", "Ballad", "Power Ballad", "Rhythmic Soul", "Freestyle",
"Duet", "Punk Rock", "Drum Solo", "A Capella", "Euro-House",
"Dance Hall", "Goa", "Drum & Bass", "Club-House", "Hardcore", "Terror",
"Indie", "Brit Pop", "Negerpunk", "Polsk Punk", "Beat",
"Christian Gangsta Rap", "Heavy Metal", "Black Metal", "Crossover",
"Contemporary Christian", "Christian Rock", "Merengue", "Salsa",
"Trash Metal", "Anime", "JPop", "Synth Pop" };
// количество стилей
#define GENRES_COUNT ( ( int) ( ( sizeof Genres) / ( sizeof
                        Genres[0])))
// частота дискретизации
static int iSampleRate[3][3] =
{
  { 32000, 16000,  8000 }, { 22050, 24000, 16000 },
  { 44100, 48000, 32000 }
};
// скорость передачи данных
static int iBitRate [6] [16] =
{
{ 0, 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 0 },
{ 0, 8, 16, 24, 32, 40, 48, 56, 64, 80, 96, 112, 128, 144, 160, 0 },
{ 0, 32, 48, 56, 64, 80, 96, 112, 128, 144, 160, 176, 192, 224, 256, 0 },
{ 0, 32, 40, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 0 },
{ 0, 32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 384, 0 },
{ 0, 32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448,
0 },
};
class CMp3Info
{
public:
   CMp3Info ();
```

```
    virtual ~CMp3Info ();
    // функция для загрузки файла MP3
    void Open ( char* szMp3File);
    // сохранить информацию в файл MP3
    void Save ( char* szMp3File);
    // функции для получения текущих значений
    int GetVersion () const; // номер версии
    int GetLayer () const; // номер уровня
    bool IsCRC () const; // защита CRC
    int GetBitrate () const; // скорость передачи данных
    int GetSampleRate () const; // частота дискретизации
    bool IsPrivate () const; // частный
    int GetMode () const; // режим
    bool IsCopyright () const; // авторское право
    bool IsOriginal () const; // оригинал
    DWORD GetSize () const; // размер файла
    void GetTitle ( char* buffer) const; // название композиции
    void GetArtist ( char* buffer) const; // имя исполнителя
    void  GetAlbum ( char* buffer) const; // название альбома
    void GetYear ( char* buffer) const; // год записи альбома
    void GetComment ( char* buffer) const; // комментарии
    void GetGenre ( char* buffer) const; // стиль музыки
    // функции для установки параметров
    void SetTitle ( char* pszTitle);
    void  SetArtist ( char* pszArtist);
    void  SetAlbum ( char* pszAlbum);
    void SetYear ( char* pszYear);
    void SetComment ( char* pszComment);
    void SetGenre ( int iGenre);
private:
    // номер версии формата
    enum MPEG_VERSION
    {
      MPEG_25, // 2.5
      MPEG_0, // резерв
      MPEG_2, // 2.0
      MPEG_1 // 1.0
    };
    // номер уровня пересмотра
    enum MPEG_LAYER
    {
      LAYER_0, // резерв
      LAYER_3, // Layer 3
```

```
    LAYER_2, // Layer 2
    LAYER_1 // Layer 1
  };
  // тип режима
  enum MPEG_MODE
  {
    STEREO, // Stereo
    JOINT_STEREO, // Joint Stereo
    DUAL_CHANNEL, // Dual Channel
    SINGLE_CHANNEL // Single Channel
  };
  // структура для хранения заголовка файла MP3
  struct MP3HEADER
  {
      int iSync;
      int iVersion;
      int iLayer;
      int iCRC;
      int iBitrate;
      int iSampleRate;
      int iPad;
      int iPrivate;
      int iMode;
      int iModeExt;
      int iCopy;
      int iOriginal;
      int Emp;
      DWORD dwSize;
  };
  MP3HEADER hdr_Mp3;
  char szTitle[30]; // название композиции
  char szArtist[30]; // имя исполнителя
  char szAlbum[30]; // название альбома
  char szYear[4]; // год записи альбома
  char szComment[30]; // комментарии
  int m_iGenre; // стиль музыки
  // функция для получения указанного параметра
  unsigned int _getValue ( DWORD dvValue, unsigned int start,
                          unsigned int len);
  char* _rightTrim ( char* str); // удаление пробелов справа в строке
}; // окончание класса CMp3Info
```

А теперь реализуем наш класс так, как показано в листинге 7.18.

Листинг 7.18. Файл CMp3Info.cpp

```
#include "stdafx.h"
#include "Mp3Info.h"
CMp3Info :: CMp3Info ()
{
  iGenre = 0;
}
CMp3Info :: ~CMp3Info ()
{
}
unsigned int CMp3Info :: getValue ( DWORD dvValue, unsigned int start,
                                    unsigned int len)
{
   return ( dvValue >> ( start - 1)) & ( ( 1 << len) - 1);
}
char* CMp3Info :: _rightTrim ( char* str)
{
   int i = strlen ( str);
   while ( ( --i) > 0 && isspace ( str [i]))
   {
     str[i] = '\0';
   }
   return ( str);
}
void CMp3Info :: Open ( char* szMp3File)
{
   HANDLE hFile = NULL;
   // открываем MP3-файл
   if ( ( hFile = CreateFile ( szMp3File, GENERIC_READ, FILE_SHARE_READ,
          NULL, OPEN_EXISTING, 0, NULL)) != INVALID_HANDLE_VALUE)
    {
     DWORD dwInfo = 0, dwReadBytes = 0, dwFileSize = 0;
     char szTemp[40];
     // получаем размер файла
     dwFileSize = GetFileSize ( hFile, NULL);
     hdr_Mp3.dwSize = dwFileSize;
     // устанавливаем маркер чтения на начало файла
     SetFilePointer ( hFile, 0, 0, FILE_BEGIN);
     // читаем заголовок файла размером 4 байта ( используются 32 бита)
     ReadFile ( hFile, &szTemp, 4, &dwReadBytes, NULL);
     // сохраняем из буфера нужные данные
     dwInfo = ( DWORD) ( ( ( szTemp[0] & 255) << 24) |
     ( ( szTemp[1] & 255) << 16) | ( ( szTemp[2] & 255) << 8) |
                                       ( ( szTemp[3] & 255)));
```

```
// распаковываем данные и сохраняем в структуру
hdr_Mp3.iSync = _getValue ( dwInfo, 22, 11);
hdr_Mp3.iVersion  = _getValue ( dwInfo, 20, 2);
hdr_Mp3.iLayer = _getValue ( dwInfo, 18, 2);
hdr_Mp3.iCRC = _getValue ( dwInfo, 17, 1);
hdr_Mp3.iBitrate = _getValue ( dwInfo, 13, 4);
hdr_Mp3.iSampleRate = _getValue ( dwInfo, 11, 2);
hdr_Mp3.iPad = _getValue ( dwInfo, 10, 1);
hdr_Mp3.iPrivate = _getValue ( dwInfo, 9,  1);
hdr_Mp3.iMode = _getValue ( dwInfo, 7,  2);
hdr_Mp3.iModeExt = _getValue ( dwInfo, 5,  2);
hdr_Mp3.iCopy = _getValue ( dwInfo, 4,  1);
hdr_Mp3.iOriginal = _getValue ( dwInfo, 3,  1);
hdr_Mp3.Emp = _getValue ( dwInfo, 1,  2);
// проверяем наличие дополнительной информации
SetFilePointer ( hFile, -128, 0, FILE_END);
szTemp[3] = '\0';
ReadFile ( hFile, szTemp, 3, &dwReadBytes, NULL);
 if ( !( strcmp ( szTemp, "TAG")))
 {
  // есть дополнительная информация
  szTemp[30] = '\0';
  // название композиции
  ReadFile ( hFile, szTemp, 30, &dwReadBytes, NULL);
  strcpy ( szTitle, _rightTrim ( szTemp));
  // имя исполнителя
  szTemp[30] = '\0';
  ReadFile ( hFile, szTemp, 30, &dwReadBytes, NULL);
  strcpy ( szArtist, _rightTrim ( szTemp));
  // название альбома
  szTemp[30] = '\0';
  ReadFile ( hFile, szTemp, 30, &dwReadBytes, NULL);
  strcpy ( szAlbum, _rightTrim ( szTemp));
  // год записи альбома
  szTemp[4] = '\0';
  ReadFile ( hFile, szTemp, 4, &dwReadBytes, NULL);
  strcpy ( szYear, _rightTrim ( szTemp));
  // комментарии
  szTemp[30] = '\0';
  ReadFile ( hFile, szTemp, 30, &dwReadBytes, NULL);
  strcpy ( szComment, _rightTrim ( szTemp));
  // стиль музыки
  m_iGenre = GENRES_COUNT + 1;
```

```
        ReadFile ( hFile, szTemp, 1, &dwReadBytes, NULL);
        m_iGenre = szTemp[0];
       }
     // закрываем файл
     CloseHandle ( hFile);
    }
}
int CMp3Info :: GetVersion ()
{
   switch ( hdr_Mp3.iVersion)
   {
    case 0: // 2.5
      return MPEG_25;
    case 1: // reserv
      return MPEG_0;
    case 2: // 2.0
      return MPEG_2;
    case 3: // 1.0.
      return MPEG_1;
   }
   return -1;
}
int CMp3Info :: GetLayer () const
{
   switch ( hdr_Mp3.iLayer )
   {
     case 0: // резерв
       return LAYER_0;
     case 1: // Layer 3
       return LAYER_3;
     case 2: // Layer 2
       return LAYER_2;
     case 3: // Layer 1
       return LAYER_1;
   }
   return -1;
}
bool CMp3Info :: IsCRC () const
{
    if ( hdr_Mp3.iCRC)
      return true;
    return false;
}
```

```
int CMp3Info :: GetBitrate () const
{
switch ( hdr_Mp3.iVersion)
  {
  case MPEG_1:
    {
     switch ( hdr_Mp3.iLayer)
            {
             case LAYER_1:
               return iBitRate[5] [hdr_Mp3.iBitrate];
             case LAYER_2:
               return iBitRate[4] [hdr_Mp3.iBitrate];
             case LAYER_3:
               return iBitRate[3] [hdr_Mp3.iBitrate];
            }
          }
  case MPEG_2:
  case MPEG_25:
   {
     switch ( hdr_Mp3.iLayer)
       {
           case LAYER_1:
               return iBitRate[2] [hdr_Mp3.iBitrate];
           case LAYER_2:
               return iBitRate[1] [hdr_Mp3.iBitrate];
           case LAYER_3:
               return iBitRate[0] [hdr_Mp3.iBitrate];
       }
   }
   return -1;
}
int CMp3Info :: GetSampleRate () const
{
switch ( hdr_Mp3.iVersion)
  {
    case MPEG_1:
      {
        switch ( hdr_Mp3.iSampleRate)
            {
              case 0:
                 return iSampleRate[3] [0];
              case 1:
                 return iSampleRate[3] [1];
```

```
            case 2:
               return iSampleRate[3] [2];
          }
      }
    case MPEG_2:
      {
        switch ( hdr_Mp3.iSampleRate)
          {
            case 0:
               return iSampleRate[2] [0];
            case 1:
               return iSampleRate[2] [1];
            case 2:
               return iSampleRate[2] [2];
          }
       }
     case MPEG_25:
       {
         switch ( hdr_Mp3.iSampleRate)
           {
             case 0:
                return iSampleRate[1] [0];
             case 1:
                return iSampleRate[1] [1];
             case 2:
                return iSampleRate[1] [2];
           }
      }
  }
   return -1;
}
bool CMp3Info :: IsPrivate () const
{
   if ( hdr_Mp3.iPrivate)
        return true;
    return false;
}
int CMp3Info :: GetMode () const
{
   switch ( hdr_Mp3.iMode )
   {
     case 0:
        return STEREO;
```

```
     case 1:
        return JOINT_STEREO;
     case 2:
        return DUAL_CHANNEL;
     case 3:
        return SINGLE_CHANNEL;
   }
   return -1;
}
bool CMp3Info :: IsCopyright () const
{
   if ( hdr_Mp3.iCopy)
        return true;
    return false;
}
bool CMp3Info :: IsOriginal() const
{
   if ( hdr_Mp3.iOriginal)
        return true;
    return false;
}
DWORD CMp3Info :: GetSize () const
{
   return hdr_Mp3.dwSize;
}
void CMp3Info :: GetTitle ( char* buffer) const
{
   strcpy ( buffer, szTitle);
}
void CMp3Info :: GetArtist ( char* buffer) const
{
   strcpy ( buffer, szArtist);
}
void CMp3Info :: GetAlbum ( char* buffer) const
{
   strcpy ( buffer, szAlbum);
}
void CMp3Info :: GetYear ( char* buffer) const
{
   strcpy ( buffer, szYear);
}
void CMp3Info :: GetComment ( char* buffer) const
{
   strcpy ( buffer, szComment);
}
```

```
void CMp3Info :: GetGenre ( char* buffer) const
{
   if ( m_iGenre > GENRES_COUNT)
     strcpy ( buffer, "Unknown");
   else
     strcpy ( buffer, Genres[m_iGenre]);
}
void CMp3Info :: SetTitle ( char* pszTitle)
{.
   if ( strlen ( pszTitle) > 30) return;
     strcpy ( szTitle, pszTitle);
}
void CMp3Info :: SetArtist ( char* pszArtist)
{
   if ( strlen ( pszArtist) > 30) return;
     strcpy ( szArtist, pszArtist);
}
void CMp3Info :: SetAlbum ( char* pszAlbum)
{
   if ( strlen ( pszAlbum) > 30) return;
     strcpy ( szAlbum, pszAlbum);
}
void CMp3Info :: SetYear ( char* pszYear)
{
   if ( strlen ( pszYear) > 4) return;
     strcpy ( szYear, pszYear);
}
void CMp3Info :: SetComment ( char* pszComment)
{
   if ( strlen ( pszComment) > 30) return;
     strcpy ( szComment, pszComment);
}
void CMp3Info :: SetGenre ( int iGenre)
{
   if ( ( iGenre < 0) && ( iGenre > GENRES_COUNT)) return;
   m_iGenre = iGenre;
}
void CMp3Info :: Save ( char* szMp3File)
{
   if ( szMp3File[0] != '\0') return;
   // открываем файл для записи
   HANDLE hFile = NULL;
   char szTemp[30];
```

```
    DWORD dwWriteBytes = 0;
    hFile = CreateFile ( szMp3File, GENERIC_WRITE, FILE_SHARE_WRITE, NULL,
                         OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
    // устанавливаем позицию для записи в файл
    SetFilePointer ( hFile, -128, 0, FILE_END);
    // записываем информационный указатель
    strcpy ( szBuffer, "TAG\0");
    WriteFile ( hFile, szTemp, 3, & dwWriteBytes, NULL);
    // название композиции
    WriteFile ( hFile, szTitle, 30, &dwWriteBytes, NULL);
    // имя испольнителя
    WriteFile( hFile, szArtist, 30, &dwWriteBytes, NULL);
    // название альбома
    WriteFile ( hFile, szAlbum, 30, &dwWriteBytes, NULL);
    // год выпуска
    WriteFile ( hFile, szYear, 4, &dwWriteBytes, NULL);
    // комментарии
    WriteFile ( hFile, szComment, 30, &dwWriteBytes, NULL);
    // стиль музыки
    szTemp[0] = ( char) m_iGenre;
    szTemp[1] = '\0';
    WriteFile ( hFile, szTemp, 1, &dwWriteBytes, NULL);
    // закрываем файл
    CloseHandle ( hFile);
}
```

Использование класса `CMp3Info` не должно вызвать никаких вопросов. Вначале объявляете его в своей программе, а далее вызываете функцию `Open`. После того как файл MP3 будет открыт, можно пользоваться остальными функциями. Перед тем как сохранить файл с помощью `Save`, необходимо заполнить информационные поля, вызывая поочередно соответствующие функции.

На этом я хотел бы завершить данную главу. Многие интересные темы остались "за бортом", но и представленная здесь информация поможет вам быстрее разобраться с программированием звука в Windows.

ГЛАВА 8

Системный динамик

Системный динамик является таким же древним устройством, как и сам X86-совместимый компьютер. Он представляет собой маленький динамик, расположенный в системном корпусе, и несколько дополнительных электронных компонентов на материнской плате. В прошлом веке, когда не было звуковых плат, именно системный динамик или, как его еще называют, спикер выполнял основные функции по извлечению звуков. Большинство старых игрушек под DOS использовали спикер для озвучивания различных игровых ситуаций: от стрельбы до звуков летящего самолета. К сожалению, качество звука оставляло желать лучшего. Потом появились отдельные устройства для воспроизведения звука, и спикер перестали применять в игровых и мультимедийных программах. Однако и по сей день на подавляющем большинстве компьютерных систем установлен маленький круглый динамик. При каждой загрузке компьютера раздается одиночный звуковой сигнал, подтверждающий успешное тестирование оборудования и выполнение операции начальной загрузки (POST). Для этого используется именно спикер, поскольку ни звуковая карта, ни любое другое устройство еще не может полноценно функционировать. Разработана целая комбинация различных звуковых кодов, позволяющих выявить сбои в подключенном оборудовании. Именно поэтому системный динамик установлен как в 486-м простеньком компьютере, так и в современном мультимедийном "монстре" на базе Pentium 4. Вы можете спросить, а зачем нужен спикер в операционных системах Windows, если там есть нормальная звуковая плата. Ответ очень прост: системный динамик гарантированно установлен на большинстве компьютеров и его можно применять для вывода системных сообщений и простых звуковых эффектов. Это, несомненно, даст вашей программе преимущество по сравнению с другими. В любом случае, вам самим решать, стоит ли использовать возможности системного динамика или нет, а я просто познакомлю вас с основными методами программирования спикера.

8.1. Программирование системного динамика

Для доступа к системному динамику используется порт с номером 61h. Он имеет размер 8 бит, но для управления спикером применяются только два младших бита (0 и 1): нулевой бит управляет включением канала 2 системного таймера, а первый бит включает динамик. Программирование системного таймера рассматривается в *гл. 10.* Установка этих битов в 1 позволяет включить динамик, а сброс — выключить. Рассмотрим пример для включения спикера (листинг 8.1).

Листинг 8.1. Включение системного динамика

```
Speaker_On proc near
push AX      ; сохраняем значение AX
in  AL, 61h  ; читаем состояние порта
or  AL, 00000011b; устанавливаем биты 0 и 1 в 1
out 61h, AL  ; включаем системный динамик
pop AX       ; восстанавливаем AX
ret
Speaker_On endp
```

Для выключения спикера можно применить код, представленный в листинге 8.2.

Листинг 8.2. Выключение системного динамика

```
Speaker_Off proc near
push AX      ; сохраняем значение AX
in  AL, 61h  ; читаем состояние порта
and AL, 11111100b; устанавливаем биты 0 и 1 в 0
out 61h, AL  ; включаем системный динамик
pop AX       ; восстанавливаем AX
ret
Speaker_Off endp
; реализуем работу спикера
call Speaker_On; включаем динамик
; выводим звук примерно 5 секунд
mov CX, 0050h ; старшее слово паузы
mov CX, 0B350h; младшее слово паузы
mov AH, 86h   ; функция BIOS паузы
int 15h       ; вызываем прерывание
call Speaker_On; включаем динамик
```

Аналогичный пример для C++ показан в листинге 8.3.

Листинг 8.3. Выключение системного динамика в C++

```
// пишем функцию управления динамиком
void Speaker ( bool bOn)
{
   DWORD dwResult = 0;
   // читаем состояние порта
   inPort ( 0x61, &dwResult, 1);
   if ( bOn) // включить динамик
   {
    dwResult |= 0x03;
    // записываем значение в порт
    outPort ( 0x61, dwResult, 1);
   }
   else // выключить динамик
   {
    dwResult &= 0xFC;
    outPort ( 0x61, dwResult, 1);
   }
}
// пишем реализацию работы спикера
Speaker ( true); // включаем динамик
delay ( 3000); // пауза 3 секунды, можно использовать функцию Sleep
Speaker ( false); // выключаем динамик
```

Рассмотренные функции управляют только включением и выключением динамика, сам же выводимый звуковой сигнал не изменяется по частоте. Чтобы немного преобразить наш звук, можно сделать так, как показано в листинге 8.4.

Листинг 8.4. Изменение частоты звука

```
mov BX, 2328h   ; определяем значение частоты
@set_tone:
call Speaker_On ; включаем динамик
mov  CX, 1388h  ; определяем значение для счетчика
@pause:
loop @pause     ; делаем паузу
call Speaker_Off; выключаем динамик
mov CX, 2328h
```

```
@pause2:
loop @pause2    ; делаем еще одну паузу
dec BX          ; уменьшаем значение в BX
jnz @set_tone   ; повторяем
```

Такой способ изменения частоты имеет серьезный недостаток — большая загрузка процессорного времени. Чтобы решить эту проблему, придется дополнительно программировать порты системного таймера. Я не буду здесь рассказывать о работе с таймером *(см. гл. 10)*, а просто приведу пример для настройки частоты выводимого звука в герцах (листинг 8.5).

Листинг 8.5. Управление частотой звука посредством системного таймера

```
SetFrequence proc near
; настраиваем регистр управления системного таймера
mov AL, 6Bh  ; канал 2, запись двух байтов, прямоугольные импульсы
out 43h, AL  ; записываем значение в управляющий порт таймера
; определяем значение частоты в 440 Гц ( 1 193 180 / 440)
mov AL, 98h  ; младший байт
out 42h, AL  ; записываем значение в порт динамика
mov AL, 0Ah  ; старший байт
out 42h, AL  ; записываем значение в порт динамика
call Speaker_On; включаем динамик
; устанавливаем паузу
mov  CX, 0C350h; определяем значение для счетчика
@pause:
loop @pause  ; делаем паузу
call Speaker_Off; выключаем динамик
ret
SetFrequence endp
```

Аналогичный пример для C++ показан в листинге 8.6.

Листинг 8.6. Управление частотой звука посредством системного таймера в C++

```
// реализуем функцию управления частотой
void SetFrequence ( unsigned int uFrequence)
{
   DWORD dwResult = 0;
   if ( uFrequence <= 0) return;
   int TimerClock = 1193180; // внутренняя частота таймера
   int iValue = 0; // значение делителя частоты
```

```
    // определяем значение делителя
    iValue = TimerClock / uFrequence;
    // настраиваем регистр управления системного таймера
    outPort ( 0x43, 0xB6, 1); // канал 2, 2 байта, прямоугольные импульсы
    dwResult = nDivider % 256; // младший байт делителя частоты
    outPort ( 0x42, dwResult, 2); // записываем значение в порт динамика
    dwResult = nDivider / 256; // старший байт делителя частоты
    outPort ( 0x42, dwResult, 2); // записываем значение в порт динамика
}
// попробуем вывести звуковой сигнал частотой примерно 1000 Гц
SetFrequence ( 1193); // устанавливаем желаемую частоту
Speaker ( true); // включаем динамик
Sleep ( 2000); // устанавливаем длительность сигнала равным 2 секунды
Speaker ( false); // выключаем динамик
```

Используя порты таймера для задания частоты, можно воспроизводить любые мелодии. Для тех, кто захочет написать иолноценное музыкальное произведение, приведу значения частот всех нот в табл. 8.1.

Таблица 8.1. *Список частот*

Нота	Значение частоты, Гц
До	4186
До-диез	4435
Ре	4699
Ре-диез	4978
Ми	5274
Фа	5588
Фа-диез	5920
Соль	6272
Соль-диез	6645
Ля	7040
Ля-диез	7459
Си	7902

Как видите, нет ничего сложного в программировании системного динамика.

ГЛАВА 9

Часы реального времени

Каждый компьютер, как минимум, ассоциируется с точностью. А точность, так или иначе, подразумевает постоянный отсчет времени. Для реализации этой задачи используется специальный модуль, называемый часами реального времени (RTC — Real Time Clock). Именно эти часы позволяют нам отслеживать дату и время, а компьютеру позволяют упорядочить свою работу, и, кроме того, два раза в год предупреждать нас о переводе стрелок часов вперед или назад. Но, как вы заметили, компьютер при необходимости выключается, а после включения все временные (и не только) параметры восстанавливаются в соответствии с текущим временем. Для этого в модуле RTC имеется небольшая микросхема памяти размером от 64 до 128 байт, выполненная по специальной технологии CMOS. Эта микросхема потребляет очень мало энергии (в качестве источника используется литиевая батарейка) и может годами сохранять постоянные значения. Поэтому сюда записывается различная системная информация, в том числе и время. После включения компьютера операционная система считывает данные из CMOS и в соответствии с ними настраивает текущие значения даты и времени.

Не буду вдаваться в детали, скажу только, что аппаратная организация RTC базируется на задающем генераторе синусоидального сигнала с частотой 32 кГц (точнее 32,768 кГц). Для подпитки применяется высококачественная батарейка (Intel рекомендует использовать элементы питания фирмы Duracell) с высокой емкостью заряда (170 мА). Для расчета времени разряда можно емкость батарейки разделить на среднее значение тока разряда (5 мкА), и мы получим значение времени, измеряемое в часах (34,000 часов или 3,88 года). Поскольку точность часов реального времени напрямую зависит от напряжения питания, следует периодически (раз в год) проверять параметры батарейки и при необходимости делать замену. Конечно, для домашних компьютеров небольшая потеря точности часов не принесет никаких проблем, но для производственных систем с компьютерным управлением, а тем более работающих в режиме реального времени, любая неточ-

ность системных часов может привести к сбою всего технологического процесса.

Для программирования часов реального времени можно воспользоваться двумя способами:

1. С помощью функций BIOS.
2. С помощью портов ввода-вывода.

Разберем каждый вариант подробнее.

9.1. Использование функций BIOS

Для поддержки RTC применяется прерывание `int 1Ah` и несколько функций. Эти функции позволяют считывать и устанавливать значения даты и времени для памяти CMOS. Прежде чем перейти к их описанию, я хотел бы предоставить вашему вниманию структуру данных в CMOS. Несмотря на полный размер памяти (от 64 до 128 байт), стандартизированы только первые 51 байт. Остальные зависят от производителя материнской платы и здесь рассматриваться не будут. Каждое смещение байта в памяти CMOS определяет номер регистра, через который можно считывать и записывать информацию. Для RTC выделены первые 13 регистров, а остальные служат для других целей. Формат микросхемы памяти представлен в табл. 9.1.

Таблица 9.1. *Формат памяти CMOS*

Адрес регистра	Описание
00h	Текущее значение секунды (00h—59h в формате BCD или 00h—3Bh)
01h	Значение секунд будильника (00h—59h в формате BCD или 00h—3Bh)
02h	Текущее значение минуты (00h—59h в формате BCD или 00h—3Bh)
03h	Значение минут будильника (00h—59h в формате BCD или 00h—3Bh)
04h	Текущее значение часа: 24-часовой режим (00h—23h в формате BCD или 00h—17h), 12-часовой режим AM (01h—12h в формате BCD или 00h—0Ch), 12-часовой режим PM (81h—92h в формате BCD или 81h—8Ch)
05h	Значение часа будильника: 24-часовой режим (00h—23h в формате BCD или 00h—17h), 12-часовой режим AM (01h—12h в формате BCD или 00h—0Ch), 12-часовой режим PM (81h—92h в формате BCD или 81h—8Ch)
06h	Текущее значение дня недели (01h—07h в формате BCD или 01h—07h, где 01h соответствует воскресенью)
07h	Текущее значение дня месяца (01h—31h в формате BCD или 01h—1Fh)
08h	Текущее значение месяца (01h—12h в формате BCD или 01h—0Ch)

Таблица 9.1 *(продолжение)*

Адрес регистра	Описание
09h	Текущее значение года (00h—99h в формате BCD или 00h—63h)
0Ah	Регистр состояния A: бит 7 — обновление времени (1 — идет обновление времени, 0 — доступ разрешен), биты 4—6 — делитель частоты (010b — 32,768 кГц), биты 0—3 — значение пересчета частоты (0110b — 1024 Гц)
0Bh	Регистр состояния B: бит 7 — запрещение обновления часов (1 — идет установка, 0 — обновление разрешено), бит 6 — разрешение периодического прерывания IRQ8 (1 — разрешено, 0 — запрещено), бит 5 — разрешение прерывания от будильника (1 — разрешено, 0 — запрещено), бит 4 — вызов прерывания после цикла обновления (1 — разрешено, 0 — запрещено), бит 3 — разрешение генерации прямоугольных импульсов (1 — разрешено, 0 — запрещено), бит 2 — выбор формата представления даты и времени (1 — двоичный, 0 — BCD), бит 1 — выбор часового режима (1 — 24 часа, 0 — 12 часов), бит 0 — автоматический переход на летнее время (1 — разрешено, 0 — запрещено)
0Ch	Регистр состояния C (только для чтения): бит 7 — признак выполненного прерывания (1 — произошло, 0 — не было), бит 6 — периодическое прерывание (1 — есть, 0 — нет), бит 5 — прерывание от будильника (1 — есть, 0 — нет), бит 4 — прерывание после обновления часов (1 — есть, 0 — нет), биты 0—3 зарезервированы и должны быть равны 0
0Dh	Регистр состояния D: бит 7 — состояние батареи и памяти (1 — память в норме, 0 — батарейка разряжена)
0Eh	Состояние POST после загрузки компьютера: бит 7 — сброс часов по питанию (1 — нет питания), бит 6 — ошибка контрольной суммы (CRT) в CMOS памяти (1 — ошибка), бит 5 — несоответствие конфигурации оборудования (1 — POST обнаружила ошибки конфигурации), бит 4 — ошибка размера памяти (1 — POST обнаружила несоответствие размера памяти с записанным в CMOS), бит 3 — сбой контроллера первого жесткого диска (1 — есть сбой), бит 2 — сбой в работе часов RTC (1 — есть сбой), биты 0 и 1 зарезервированы и должны быть равны 0
0Fh	Состояние компьютера перед последней загрузкой: 00h — был выполнен сброс по питанию (кнопка Reset или <Ctrl>+<Alt>+<Del>), 03h — ошибка тестирования памяти, 04h — POST была завершена и система перезагружена, 05h — переход (jmp dword ptr) на адрес в 0040h:0067h, 07h — ошибка защиты при тестировании, 08h — ошибка размера памяти
10h	Тип дисковода (0—3 для A и 4—7 для B): 0000b — нет, 0001b — 360 Кб, 0010b — 1,2 Мб, 0011b — 720 Кб, 0100b — 1,44 Мб, 0101b — 2,88 Мб
11h	Резерв
12h	Тип жесткого диска (0—3 для D и 4—7 для C): 0000b — нет, 0001b—1110b — тип дисковода (от 1 до 14), 1111b — диск первый описывается в регистре 19h, а диск второй — в 1Ah

Таблица 9.1 *(окончание)*

Адрес регистра	Описание
13h	Резерв
14h	Состояние оборудования: биты 6—7 — число флоппи-дисководов (00b — один, 01b — два), биты 4—5 — тип дисплея (00b — EGA или VGA, 01b — цветной 40 × 25, 10b — цветной 80 × 25, 11b — монохромный 80 × 25), биты 2 и 3 зарезервированы, бит 1 — наличие сопроцессора (1 — есть), бит 0 — наличие флоппи-дисковода (1 — есть)
15h	Младший байт размера основной памяти в килобайтах (80h)
16h	Старший байт размера основной памяти в килобайтах (02h)
17h	Младший байт размера дополнительной памяти в килобайтах
18h	Старший байт размера дополнительной памяти в килобайтах
19h	Тип первого жесткого диска
1Ah	Тип второго жесткого диска
1Bh—2Dh	Резерв
2Eh	Старший байт контрольной суммы регистров CMOS (10h—2Dh)
2Fh	Младший байт контрольной суммы регистров CMOS (10h—2Dh)
30h	Младший байт размера дополнительной памяти в килобайтах
31h	Старший байт размера дополнительной памяти в килобайтах
32h	Значение века в формате BCD
33h	Дополнительный флаг свойств: бит 7 — размер памяти (1 — больше 1 Мб, 0 — до 1 Мб), бит 6 — используется программой установки, биты 0—5 зарезервированы
34h—3Fh (7Fh)	Зависят от производителя

Используемый формат представления данных BCD (Binary Coded Decimal) представляет собой двоично-десятичный код, где каждый байт содержит два независимых значения. Первое из них кодируется битами 7—4, а второе — битами 3—0. Поддерживаются только положительные значения до 99 включительно.

А теперь рассмотрим функции BIOS для управления часами реального времени.

9.1.1. Функция *00h*

Данная функция позволяет получить текущее значение счетчика RTC.

Использование:

1. В регистр AH следует поместить код функции 00h.
2. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AL будет записан код перехода: 00h — произошел переход времени через полночь. В регистры CX и DX будут записаны соответственно старшее и младшее значения счетчика. В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. Поскольку частота сигнала прерывания составляет 18,2 Гц (18,2 прерываний в секунду), за сутки набегает немногим более 1 572 480 отсчетов, а после этого счетчик сбрасывается в 0. Например, чтобы получить текущее значение счетчика, можно использовать код из листинга 9.1.

Листинг 9.1. Получение текущего значения счетчика

```
; определяем переменную для хранения полученного значения
Time_Count dd ?
; общий код программы
mov AH, 00h   ; код функции 00h
int 1Ah       ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
; сохраняем текущее значение счетчика
mov word ptr Time_Count, DX; младшее слово
mov word ptr Time_Count + 2, CX; младшее слово
```

9.1.2. Функция *01h*

Функция позволяет установить новое значение счетчика для системного таймера.

Использование:

1. В регистр AH следует поместить код функции 01h.
2. В регистр CX следует записать старшее слово значения счетчика.
3. В регистр DX следует записать младшее слово значения счетчика.
4. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен.

9.1.3. Функция *02h*

Эта функция позволяет получить текущее значение времени в формате BCD.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. При успешном выполнении функции в регистры будут записаны следующие значения: CH — значение часа, CL — значение минуты, DH — значение секунды, DL — значение перехода на летнее время (1 — есть переход, 0 — нет). Посмотрите в листинге 9.2, как можно получить текущее время в формате BCD.

Листинг 9.2. Получение текущего времени в формате BCD

```
; определяем переменные для хранения полученных значений
Current_Hour db ?
Current_Minute db ?
Current_Second db ?
; общий код программы
mov AH, 02h   ; код функции 02h
int 1Ah       ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
; сохраняем текущее значение времени
mov byte ptr Current_Hour, CH; часы
mov byte ptr Current_Minute, CL; минуты
mov byte ptr Current_Second, DH; секунды
```

9.1.4. Функция *03h*

Функция позволяет установить текущее значение времени в формате BCD.

Использование:

1. В регистр AH следует поместить код функции 03h.
2. В регистр CH следует записать часы.
3. В регистр CL следует записать минуты.
4. В регистр DH следует записать секунды.
5. В регистр DL нужно поместить значение перехода на летнее время.
6. Вызвать прерывание int 1Ah.

Выход:

В случае ошибки флаг переноса `CF` будет установлен в 1, иначе сброшен. Пример для установки нового системного времени приведен в листинге 9.3.

Листинг 9.3. Установка нового системного времени

```
mov CH, 11h    ; 11 часов
mov CL, 15h    ; 15 минут
mov DH, 30h    ; 30 секунд
mov DL, 1      ; с переходом на летнее время
mov AH, 03h    ; код функции 03h
int 1Ah        ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
```

9.1.5. Функция *04h*

Эта функция позволяет получить текущую дату в формате BCD.

Использование:

1. В регистр `AH` следует поместить код функции `04h`.
2. Вызвать прерывание `int 1Ah`.

Выход:

В случае ошибки флаг переноса `CF` будет установлен в 1, иначе сброшен. При успешном выполнении функции в регистры будут записаны следующие значения: `CH` — значение века, `CL` — значение года, `DH` — значение месяца, `DL` — значение дня. Например, для получения текущей даты можно использовать код, представленный в листинге 9.4.

Листинг 9.4. Получение текущей даты в формате BCD

```
; определяем переменные для хранения полученных значений
Current_Vek db ?
Current_Year db ?
Current_Month db ?
Current_Day db ?
; общий код программы
mov AH, 04h    ; код функции 04h
int 1Ah        ; вызываем прерывание
jc  ERROR_HND ; если произошла ошибка, передаем управление обработчику
; сохраняем текущее значение даты
mov byte ptr Current_Vek, CH; век
```

```
mov byte ptr Current_Year, CL; год
mov byte ptr Current_Month, DH; месяц
mov byte ptr Current_Day, DH; день
```

9.1.6. Функция *05h*

Данная функция позволяет установить текущую дату в формате BCD.

Использование:

1. В регистр `AH` следует поместить код функции `05h`.
2. В регистр `CH` следует записать век.
3. В регистр `CL` следует записать год.
4. В регистр `DH` следует записать месяц.
5. В регистр `DL` нужно поместить значение дня.
6. Вызвать прерывание `int 1Ah`.

Выход:

В случае ошибки флаг переноса `CF` будет установлен в 1, иначе сброшен.

9.1.7. Функция *06h*

Функция позволяет установить время срабатывания для будильника.

Использование:

1. В регистр `AH` следует поместить код функции `06h`.
2. В регистр `CH` следует записать часы.
3. В регистр `CL` следует записать минуты.
4. В регистр `DH` следует записать секунды.
5. Вызвать прерывание `int 1Ah`.

Выход:

В случае ошибки флаг переноса `CF` будет установлен в 1, иначе сброшен.

9.1.8. Функция *07h*

Эта функция позволяет выключить будильник.

Использование:

1. В регистр `AH` следует поместить код функции `07h`.
2. Вызвать прерывание `int 1Ah`.

Выход:

В случае ошибки флаг переноса CF будет установлен в 1, иначе сброшен. При успешном выполнении в регистр AL будет записано значение регистра CMOS по адресу 0Bh.

9.2. Использование портов

Для доступа к часам реального времени используются всего два порта: 70h и 71h. Порт 70h используется только для записи и позволяет выбрать адрес регистра в CMOS памяти. Порт 71h применяется как для записи, так и для чтения данных из указанного (через порт 70h) регистра CMOS. Оба порта являются 8-разрядными. Кроме того, бит 7 в порту 70h не относится к работе с RTC, а управляет режимом немаскируемых прерываний (1 — прерывания запрещены). Если у вас будут проблемы с доступом к регистрам CMOS, следует запрещать прерывания (команда cli) перед началом работы и разрешать прерывания (команда sti) после. Но, как правило, этого не требуется. Пример получения числа установленных флоппи-дисководов показан в листинге 9.5.

Листинг 9.5. Определение числа установленных флоппи-дисководов

```
mov AL, 14h   ; регистр CMOS 14h
out 70h, AL   ; записываем значение в порт
jmp $ + 2     ; делаем небольшую паузу
in  AL, 71h   ; читаем значение регистра
test AL, 0    ; проверяем наличие дисковода в системе
jz  NO_FDD    ; дисководов нет
and AL, 0C0h ; выделяем значение в битах 6 и 7
shr AL, 6     ; получаем результат
; сохраняем количество дисководов в переменную
cmp AL, 0     ; один дисковод
jne @dalee
mov byte ptr NUM_FDD, 1
@dalee:
cmp AL, 01    ; два дисковода
jne ERROR_HND; не удалось определить число дисководов
mov byte ptr NUM_FDD, 2
```

Аналогичный пример для C++ показан в листинге 9.6.

Листинг 9.6. Определение числа установленных флоппи-дисководов в C++

```
// напишем функцию для определения числа флоппи-дисководов
int GetNumFDD ()
```

```
{
   DWORD dwResult = 0;
   // записываем значение регистра CMOS 14h в порт
   outPort ( 0x70, 0x14, 1);
   // делаем небольшую паузу
   Sleep ( 1);
   // читаем значение из порта 71h
   inPort ( 0x71, &dwResult, 1);
   // проверяем наличие дисковода в системе
   if ( ( dwResult & 0x01) == 0x00) return 0;
   // выделяем значение в битах 6 и 7
   dwResult &= 0x0C;
   dwResult = dwResult >> 6; // выделяем результат
   // проверяем
   switch ( dwResult)
   {
    case 0:
     return 1; // один флоппи-дисковод
    case 1:
     return 2; // два флоппи-дисковода
   }
   return 0;
}
```

Таким же способом можно получить или записать любое значение в регистры CMOS. Давайте попробуем выполнить очистку памяти CMOS. Хотя эта операция и позволяет осуществить программный сброс памяти, использовать ее стоит очень осторожно. Желательно, чтобы на материнской плате были установлены две микросхемы CMOS. Причина в том, что на старых и некачественных платах затирание ячеек памяти может привести к полному отказу начальной загрузки компьютера. Это может быть связано и с изношенностью микросхемы, и с плохим качеством исполнения, а может зависеть от батарейки. На современных качественных платах таких проблем, как правило, не возникает, но возможность нарушения работы системы все равно существует. Исходя из этого, я приведу пример очистки памяти CMOS, но ответственность за любые проблемы будет лежать целиком на вас. Итак, рассмотрим следующий пример, показанный в листинге 9.7.

Листинг 9.7. Очистка памяти CMOS

```
Clear_CMOS proc near
xor CX, CX   ; обнуляем регистр CX
mov CL, 3Fh  ; предполагаем, что размер CMOS равен 64 байта
```

```
@repeat:
mov AL, CL    ; передаем номер регистра CMOS
out 70h, AL   ; записываем значение в порт
nop           ; делаем небольшую паузу
out 71h, AL   ; записываем значение в порт
loop @repeat ; повторяем для следующего регистра
Clear_CMOS endp
```

Для очистки памяти CMOS в C++ используйте код из листинга 9.8.

Листинг 9.8. Очистка памяти CMOS в C++

```
// напишем функцию для очистки CMOS
void Clear_CMOS ( unsigned int uSizeCMOS)
{
   if ( uSizeCMOS < 64) return;
   for ( int i = 0; i < uSizeCMOS; i++)
   {
     outPort ( 0x70, i, 1); // записываем номер регистра в CMOS
     outPort ( 0x71, i, 1); // записываем значение в регистр
   }
}
// воспользуемся нашей функцией для очистки памяти CMOS размером 64 байта
Clear_CMOS ( 64);
```

И напоследок я хочу привести примеры функций для C++, позволяющие писать (листинг 9.9) и читать (листинг 9.10) память CMOS в файл.

Листинг 9.9. Сохранение памяти CMOS в файл

```
// пишем функцию сохранения CMOS в файл
 bool Save_CMOS ( char* FileName)
{
   FILE* outFile;
   DWORD dwResult = 0;
   BYTE buffer[64];
   // открываем новый файл
   if ( ( outFile = fopen ( FileName, "wb")) == NULL)
      return false; // не удалось создать файл
   for ( int i = 0; i < 64; i++)
   {
     outPort ( 0x70, i, 1); // записываем номер регистра в CMOS
     Sleep ( 1); // пауза
```

```
    inPort ( 0x71, &dwResult, 1); // читаем байт из CMOS
    buffer[i] = dwResult;
   }
   // записываем данные в файл
   fwrite ( buffer, 1, 64, outFile);
   // закрываем файл
   fclose( outFile);
   return true;
}
```

Листинг 9.10. Загрузка памяти CMOS из файла

```
// пишем функцию загрузки CMOS из файла
bool Load_CMOS ( char* FileName)
{
   FILE* inFile;
   DWORD dwResult = 0;
   BYTE buffer[64];
   // открываем новый файл
   if ( ( inFile = fopen ( FileName, "wb")) == NULL)
      return false; // не удалось создать файл
   // читаем файл в буфер
   fread( buffer, 1, 64, inFile);
   // закрываем файл
   fclose( inFile);
   // сохраняем данные в CMOS
   for ( int i = 0; i < 64; i++)
   {
    dwResult = buffer[i];
    outPort ( 0x70, i, 1); // записываем номер регистра в CMOS
    outPort ( 0x71, dwResult, 1); // записываем значение в регистр
   }
   return true;
}
```

На этом можно завершить программирование часов реального времени и памяти CMOS.

ГЛАВА 10

Таймер

В любом компьютере наряду с часами реального времени существует устройство системного таймера, для реализации которого применяется специальная микросхема (например, 8254). Он помогает организовать всевозможные временные задержки, счетчики и управляющие сигналы. Из всех предоставляемых таймером функций, можно выделить несколько основных:

1. Организация часов реального времени.
2. Программируемый генератор прямоугольных и синусоидальных импульсов.
3. Счетчик событий таймера.
4. Управление двигателями флоппи-дисководов.

Таймер предоставляет три независимых канала, каждый из которых имеет свое назначение. В первом канале (0) отслеживается текущее значение времени от момента включения компьютера, для хранения которого используется область памяти BIOS (`0040:006C`). Каждое изменение значения (18,2 раз в секунду) в этом канале генерирует прерывание `IRQ0` (`int 8h`). Данное прерывание будет обработано процессором в первую очередь, но при этом должны быть разрешены аппаратные прерывания. При программировании первого канала всегда следует после выполнения задачи восстанавливать его первоначальное состояние. Второй канал (1) используется системой для работы с контроллером прямого доступа к памяти (DMA). Третий канал (2) позволяет управлять системным динамиком.

Генератор сигналов таймера вырабатывает импульсы с частотой 1 193 180 Гц. Поскольку максимальное значение 16-битного регистра ограничено значением 65 535, используется делитель частоты. В итоге, результирующее значение равно 18,2 Гц. Именно с такой частотой выдается прерывание `IRQ0`.

Для работы с системным таймером используются порты от `40h` до `43h`. Все они имеют размер 8 бит. Порты с номерами `40h`, `41h` и `42h` связаны соответ-

ственно с первым, вторым и третьим каналами, а порт 43h работает с управляющим регистром таймера. Принцип работы с портами очень простой: в порт 43h записывается управляющая команда, а после этого данные записываются или считываются из 40h, 41h и 42h, в зависимости от решаемой задачи. Формат командного регистра представлен в табл. 10.1.

Таблица 10.1. *Формат управляющего регистра таймера*

Биты	7	6	5	4	3	2	1	0
Описание	Номер канала		Тип операции		Режим		Формат	

Приведем краткое описание табл. 10.1.

- Бит 0 определяет формат представления данных: 0 — двоичный (16-битное значение от 0000h до FFFFh), 1 — двоично-десятичный (BCD от 0000 до 9999).
- Биты 1—3 определяют режим работы таймера. Существуют шесть режимов работы (от 0 до 5). Возможные значения перечислены в табл. 10.2.
- Биты 4—5 определяют тип операции. Имеются четыре возможных значения, которые перечислены в табл. 10.3.
- Биты 6—7 позволяют выбрать номер канала или управляющий регистр (только для операций чтения). Возможные значения этого поля перечислены в табл. 10.4.

Таблица 10.2. *Режимы работы таймера*

Бит 3	Бит 2	Бит 1	Описание режима
0	0	0	Генерация прерывания IRQ0 при установке счетчика в 0
0	0	1	Установка в режим ждущего мультивибратора
0	1	0	Установка в режим генератора импульсов
0	1	1	Установка в режим генератора прямоугольных импульсов
1	0	0	Установка в режим программно-зависимого одновибратора
1	0	1	Установка в режим аппаратно-зависимого одновибратора

Таблица 10.3. *Тип операции*

Бит 5	Бит 4	Тип операции
0	0	Команда блокировки счетчика
0	1	Чтение/запись только младшего байта

Таблица 10.3 (окончание)

Бит 5	Бит 4	Тип операции
1	0	Чтение/запись только старшего байта
1	1	Чтение/запись младшего, а за ним старшего байта

Таблица 10.4. Возможные значения для поля 6–7

Бит 7	Бит 6	Описание
0	0	Выбор первого канала (0)
0	1	Выбор второго канала (1)
1	0	Выбор третьего канала (2)
1	1	Команда считывания значений из регистров каналов

При установке битов 6 и 7 управляющего регистра в 1 будет использована команда считывания данных. При этом формат определения этого регистра меняется согласно табл. 10.5.

Таблица 10.5. Формат управляющего регистра в режиме считывания

Биты	7	6	5	4	3	2	1	0
Описание	1	1	Б	Статус	Канал 2	Канал 1	Канал 0	0

Приведем краткое описание таблицы.

- Бит 0 не используется и должен быть установлен в 0.
- Биты 1—3 позволяют установить номера каналов. Установка бита в 1 выбирает соответствующий номер канала.
- Бит 4 позволяет получить состояние для выбранного канала (каналов) при установке в 0.
- Бит 5 позволяет зафиксировать значение счетчика для выбранного канала (каналов) при установке в 0.
- Биты 6 и 7 определяют команду чтения и должны быть установлены в 1.

При выполнении команды блокировки счетчика (биты 4 и 5 установлены в 0) можно получить значение (состояние) выбранного счетчика без остановки самого таймера. Результат считывается из указанного канала (биты 6 и 7). При этом формат команды будет таким, как показано в табл. 10.6.

Таблица 10.6. *Формат команды блокировки*

Биты	7	6	5	4	3	2	1	0
Описание	Номер канала		0		0		Не используются	

Приведем краткий комментарий к таблице.

- Биты 0—3 не используются и должны игнорироваться.
- Биты 4 и 5 определяют команду блокировки и должны быть установлены в 0.
- Биты 6 и 7 определяют номер канала (см. табл. 10.4), для которого будет выполнена команда блокировки.

Полученный байт состояния имеет определенный формат (табл. 10.7).

Таблица 10.7. *Формат байта состояния канала*

Биты	7	6	5	4	3	2	1	0
Описание	OUT	Готов	Тип операции		Режим работы			Формат

Приведем краткое пояснение к таблице 10.7.

- Бит 0 определяет формат представления данных: 0 — двоичный (16-битное значение от `0000h` до `FFFFh`), 1 — двоично-десятичный (BCD от `0000` до `9999`).
- Биты 1—3 определяют режим работы таймера. Возможные значения перечислены в табл. 10.2.
- Биты 4—5 определяют тип операции. Возможные значения перечислены в табл. 10.3.
- Бит 6 определяет готовность счетчика для считывания данных (1 — готов, 0 — счетчик обнулен).
- Бит 7 определяет состояние выходного сигнала на канале в момент блокировки счетчика импульсов.

Рассмотрим стандартный пример для установки начального значения счетчика для второго канала (управляет динамиком) равным 5 120 Гц (листинг 10.1).

Листинг 10.1. Установка начального значения счетчика для второго канала

```
mov AL, 10110110b   ; канал 2, операция 4, режим 3, формат 0
out 43h, AL  ; записываем значение в порт
mov AX, 0E9  ; определяем делитель частоты для получения 5120 Гц
```

```
out 42h, AL  ; посылаем младший байт делителя
mov AL, AH   ; копируем значение
out 42h, AL  ; посылаем старший байт делителя
```

Аналогичный пример для C++ показан в листинге 10.2.

Листинг 10.2. Установка начального значения счетчика для второго канала в C++

```
// пишем функцию установки значения счетчика
void SetCount ( int iDivider)
{
   int iValue = 0;
   outPort ( 0x43, 0xB6, 1); // канал 2, операция 4, режим 3, формат 0
   iValue = iDivider & 0x0F;
   outPort ( 0x42, iValue, 1); // младший байт делителя
   outPort ( 0x42, ( iDivider >> 4), 1); // старший байт делителя
}
```

А теперь рассмотрим пример для получения случайного значения в установленном диапазоне (листинг 10.3).

Листинг 10.3. Получение случайного значения в диапазоне от 0 до 99

```
; выделяем место для переменной
Random_Value db ?
; сначала пишем процедуру для установки режима работы таймера
TimerInit proc near
mov AL, 10110111b   ; канал 2, операция 4, режим 3, формат 1
out 43h, AL  ; записываем значение в порт
mov AX, 11931; определяем делитель частоты
out 42h, AL  ; посылаем младший байт делителя
mov AL, AH   ; копируем значение
out 42h, AL  ; посылаем старший байт делителя
ret
TimerInit endp
; пишем процедуру получения случайного значения
GetRandomValue proc near
mov AL, 10000000b   ; канал 2, получить значение, биты 3-0 игнорируем
out 43h, AL  ; записываем значение в порт
in  AL, 42h  ; читаем младший байт значения счетчика
mov AH, AL   ; копируем его в AH
in  AL, 42h  ; читаем старший байт значения счетчика
```

```
call TimerInit      ; устанавливаем таймер заново
xchg AH, AL  ; меняем местами младший и старший байты
; сохраняем полученное значение в переменную
mov byte ptr Random_Value, AX
ret
GetRandomValue endp
```

Реализация генератора случайных чисел на C++ представлена в листинге 10.4.

Листинг 10.4. Получение случайного значения в C++

```
// пишем функцию установки таймера
void InitCount ( int iMaximum)
{
   int iValue = 0;
   int iDiv = 1193180 / iMaximum;
   outPort ( 0x43, 0xB7, 1); // канал 2, операция 4, режим 3, формат 1
   iValue = iDiv & 0x0F;
   outPort ( 0x42, iValue, 1); // младший байт делителя
   outPort ( 0x42, ( iDiv >> 4), 1); // старший байт делителя
}
// пишем функцию генерации случайного значения
int GetRandomValue ( int iMaximum)
{
   DWORD dwLSB = 0, dwMSB = 0;
   // канал 2, получить значение, биты 3-0 игнорируем
   outPort ( 0x43, 0x80, 1);
   // читаем младший байт значения счетчика
   inPort ( 0x42, &dwLSB, 1);
   // устанавливаем таймер заново
   InitCount ( iMaximum);
   // читаем старший байт значения счетчика
   inPort ( 0x42, &dwMSB, 1);
   return ( int) ( ( WORD) ( ( ( BYTE) ( dwLSB)) |
          ( ( ( WORD) ( ( BYTE) ( dwMSB))) << 8)));
}
```

На этом можно считать тему программирования системного таймера завершенной.

ГЛАВА 11

Дисковая подсистема

Каждый день мы садимся за компьютер и выполняем различную работу: кто-то заполняет бухгалтерские бланки, кто-то набирает текст, а кто-то скачивает музыку из Интернета. Что бы вы ни делали, постоянно приходится, так или иначе, сохранять результаты своего труда. Для этого используются наиболее популярные устройства: жесткий диск, гибкий диск или компакт-диск. Поэтому любой серьезный программист должен уметь работать с ними, несмотря на постоянное упрощение языков программирования. В этой главе мы научимся управлять данными устройствами как минимум тремя разными способами:

1. С использованием функций BIOS.
2. С использованием портов ввода-вывода.
3. С помощью интерфейса Win32 API.

Каждый способ имеет свои плюсы и минусы, а также определенную необходимость применения в той или иной ситуации. Окончательный выбор будет зависеть только от вас. И еще одно важное замечание: неправильное использование функций и других средств доступа к дискам может привести к полной или частичной потере информации, а также к поломке самого устройства.

11.1. Использование функций BIOS

Функции BIOS для поддержки дисковой подсистемы можно разделить на две группы: стандартные и дополнительные. Первая группа функций рассчитана на работу с дисками размером до 528 Мбайт и использует адресацию в формате CHS (cylinder-head-sector — цилиндр-головка-сектор). Все современные диски используют логическую адресацию, но могут эмулировать и CHS. С помощью функций BIOS можно получить прямой доступ к структуре носителя, что может потребоваться при разработке специальных

дисковых утилит (например, программы для восстановления удаленных файлов). Следует помнить, что функции BIOS работают с физическим устройством диска, а не с логическими разделами. Все функции BIOS для работы с дисками используют прерывание `int 13h`. Рассмотрим их подробнее.

11.1.1. Функция *00h*

Данная функция предназначена для сброса дискового устройства, которое заключается в перекалибровке головок диска и перевода их в исходную позицию.

Использование:

1. В регистр `AH` следует поместить код функции `00h`.
2. В регистр `DL` следует поместить номер дисковода. Номера жестких дисков лежат в диапазоне `80h—FFh`, а гибких — `00h—7Fh`. Установка бита 7 в 1 позволит сразу выполнить сброс всех доступных дисководов.
3. Вызвать прерывание `int 13h`.

Выход:

После выполнения функции в регистр `AH` будет помещен код состояния. Флаг `CF` в случае ошибки будет установлен в 1, иначе сброшен. Возможные коды ошибок представлены в табл. 11.1.

Таблица 11.1. *Коды состояния*

Код	Описание
00h	Операция успешно завершена
01h	Недопустимый код функции или значение параметра функции
02h	Адресный указатель не найден
03h	Диск защищен от записи
04h	Ошибка чтения, не найден сектор
05h	Сброс устройства не был выполнен (только для жесткого диска)
06h	Смена диска (только для флоппи-дисковода)
07h	Ошибка параметра диска (только для жесткого диска)
08h	Переполнение DMA
09h	Превышение границы в 64 Кб для DMA
0Ah	Обнаружен сбойный сектор (только для жесткого диска)
0Bh	Обнаружена сбойная дорожка (только для жесткого диска)
0Ch	Несовместимый тип дорожки или недопустимый носитель

Таблица 11.1 *(окончание)*

Код	Описание
0Dh	Недопустимое количество секторов для указанного формата
0Eh	Обнаружен адрес указателя на управляющие данные (только для жесткого диска)
0Fh	Превышения диапазона для уровня арбитража DMA (только для жесткого диска)
10h	Ошибка чтения (по коду CRC или ECC)
11h	Ошибка в данных исправлена по коду избыточности ECC (только для жесткого диска)
20h	Сбой контроллера
31h	Отсутствует носитель
32h	CMOS содержит неправильный тип установленного дисковода
40h	Ошибка поиска
80h	Диск не готов
AAh	Диск не готов (только для жесткого диска)
B0h	Том не заблокирован в дисководе
B1h	Том заблокирован в дисководе
B2h	Том нельзя извлечь из дисковода
B3h	Том в настоящий момент используется
B4h	Превышено значение счетчика блокировок
B5h	Не удалось извлечь носитель
B6h	Том присутствует, но доступен только для чтения
BBh	Неопределенная ошибка (только для жесткого диска)
CCh	Ошибка записи (только для жесткого диска)
E0h	Ошибка в регистре состояния
FFh	Ошибка выполнения операции статуса (только для жесткого диска)

Рассмотрим простой пример для сброса флоппи-диска, показанный в листинге 11.1.

Листинг 11.1. Сброс флоппи-дисковода

```
mov AH, 00h  ; функция сброса
mov DL, 00h  ; номер первого дисковода
int 13h      ; вызываем прерывание
```

11.1.2. Функция *01h*

Данная функция предназначена для получения информации о статусе последней выполненной команды.

Использование:

1. В регистр AH следует поместить код функции 01h.
2. В регистр DL следует поместить номер дисковода. Номера жестких дисков лежат в диапазоне 80h—FFh, а номера гибких — 00h—7Fh.
3. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. В листинге 11.2 показан пример использования этой функции.

Листинг 11.2. Сброс первого жесткого диска

```
; сначала выполним сброс первого жесткого диска
mov AH, 00h  ; функция сброса
mov DL, 80h  ; номер первого дисковода
int 13h      ; вызываем прерывание
; теперь проверим результат операции сброса
mov AH, 01h  ; функция 01h
mov DL, 80h  ; тот же дисковод
int 13h      ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

11.1.3. Функция *02h*

Функция позволяет прочитать один или более секторов с указанного диска и поместить в указанное место в памяти. Для DOS и большинства Windows размер сектора равен 512 байт.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. В регистр AL следует записать количество секторов, которые должны быть прочитаны. Это значение не должно быть меньше 1.
3. В регистр CH записываются младшие 8 битов номера цилиндра. Номер цилиндра для жестких дисков имеет размер 10 бит. Два старших бита (8 и 9) определяются в регистре CL (6 и 7 биты).
4. В регистр CL записывается номер начального сектора от 1 до 63 (используются биты 0—5). Для жестких дисков в биты 6 и 7 следует записать значения старших битов для номера цилиндра.

5. В регистр DH нужно записать номер головки.
6. В регистр DL следует поместить номер дисковода. Номера жестких дисков лежат в диапазоне 80h—FFh, а гибких — 00h—7Fh.
7. В ES:BX нужно поместить указатель на буфер данных.
8. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. В регистр AL будет записано количество реально прочитанных секторов. Рассмотрим пример кода для чтения одного сектора с жесткого диска, показанный в листинге 11.3.

Листинг 11.3. Чтение сектора жесткого диска

```
; сначала выделим память под буфер
data_read db 512 DUP (?)
; различный общий код
mov AH, 02h  ; функция 02h
mov AL, 01h  ; один сектор
mov BX, offset data_read; указатель на буфер
mov CH, 00h  ; номер цилиндра
mov CL, 00h  ; номер начального сектора
mov DH, 00h  ; номер головки
mov DL, 80h  ; первый жесткий диск
int 13h      ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

11.1.4. Функция *03h*

Эта функция предназначена для записи одного или более секторов на указанный диск.

Использование:

1. В регистр AH следует поместить код функции 03h.
2. В регистр AL следует записать количество секторов, которые должны быть записаны. Это значение не должно быть меньше 1.
3. В регистр CH записываются младшие 8 битов номера цилиндра. Номер цилиндра для жестких дисков имеет размер 10 бит. Два старших бита (8 и 9) определяются в регистре CL (6 и 7 биты).
4. В регистр CL записываются номер начального сектора от 1 до 63 (используются биты 0—5). Для жестких дисков в биты 6 и 7 следует записать значения старших битов для номера цилиндра.

5. В регистр DH нужно записать номер головки.
6. В регистр DL следует поместить номер дисковода. Номера жестких дисков лежат в диапазоне 80h—FFh, а номера гибких — 00h—7Fh.
7. В ES:BX нужно поместить указатель на буфер данных.
8. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. В регистр AL будет записано количество реально записанных секторов. В листинге 11.4 показан пример кода для записи одного сектора на жесткий диск.

Листинг 11.4. Запись сектора на жесткий диск

```
; сначала выделим память под буфер
data_write db 512 DUP (0)
; различный общий код
mov AH, 03h   ; функция 03h
mov AL, 01h   ; один сектор
mov BX, offset data_write; указатель на буфер
mov CH, 00h   ; номер цилиндра
mov CL, 00h   ; номер начального сектора
mov DH, 00h   ; номер головки
mov DL, 80h   ; первый жесткий диск
int 13h       ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

Ни в коем случае не пытайтесь выполнить приведенный выше код! Пример показан исключительно для демонстрационных целей. Игнорирование этого условия приведет к затиранию загрузочной области вашего жесткого диска и невозможности дальнейшей работы. По умолчанию, загрузочный сектор будет перезаписан пустыми значениями, и вы потеряете всю информацию!

11.1.5. Функция *04h*

Функция предназначена для проверки записанного сектора или секторов на указанном диске. Она позволяет контролировать процесс записи данных посредством считывания и проверки контрольной суммы.

Использование:

1. В регистр AH следует поместить код функции 04h.
2. В регистр AL следует записать количество секторов, которые должны быть проверены. Это значение не должно быть меньше 1.

3. В регистр CH записываются младшие 8 битов номера цилиндра. Номер цилиндра для жестких дисков имеет размер 10 бит. Два старших бита (8 и 9) определяются в регистре CL (6 и 7 биты).
4. В регистр CL записывается номер начального сектора от 1 до 63 (используются биты 0—5). Для жестких дисков в биты 6 и 7 следует записать значения старших битов для номера цилиндра.
5. В регистр DH нужно записать номер головки.
6. В регистр DL следует поместить номер дисковода. Номера жестких дисков лежат в диапазоне 80h—FFh, а гибких — 00h—7Fh.
7. В ES:BX нужно поместить указатель на буфер данных.
8. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. В регистр AL будет записано количество реально проверенных секторов.

11.1.6. Функция *05h*

Функция предназначена для форматирования дорожки гибкого диска (дискеты). Функцию можно использовать только для флоппи-дисководов.

Использование:

1. В регистр AH следует поместить код функции 05h.
2. В регистр AL нужно записать количество секторов.
3. В регистр CH нужно записать номер дорожки.
4. В регистр DH нужно записать номер головки.
5. В регистр DL следует поместить номер дисковода. Номера гибких дисков лежат в диапазоне 00h—7Fh.
6. В ES:BX нужно поместить указатель на адрес буфера, содержащего список полей для форматирования. Размер каждого поля равен 4 байта, а формат показан в табл. 11.2.
7. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен.

Таблица 11.2. Формат поля

Смещение адреса	Размер	Описание
00h	Байт	Номер дорожки
01h	Байт	Номер головки (начиная с 0)
02h	Байт	Номер сектора
03h	Байт	Размер сектора (00h — 128 байт, 01h — 256 байт, 02h — 512 байт, 03h — 1024 байта)

11.1.7. Функция *08h*

Эта функция позволяет получить информацию о дисководе.

Использование:

1. В регистр AH следует поместить код функции 08h.
2. В регистр DL следует поместить номер дисковода. Номера жестких дисков лежат в диапазоне 80h—FFh, а номера гибких — 00h—7Fh.
3. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. В регистр BL будет записан тип дисковода (табл. 11.3) для гибких дисков. В CH будут записаны младшие 8 битов максимального номера цилиндра, а в CL — максимальный номер сектора (биты 0—5) и два старших бита (6 и 7) номера цилиндра. В регистр DH будет помещен максимальный номер головки, а в DL — общее количество дисководов. Кроме того, для флоппи-дисков по адресу ES:DI будет помещен указатель на список параметров.

Таблица 11.3. Типы дисководов

Код	Описание
01h	360 Кб
02h	1,2 Мб
03h	720 Кб
04h	1,44 Мб
06h	2,88 Мб
10h	Устройство ATAPI с поддержкой сменных носителей

Рассмотрим пример для проверки наличия стандартного 3-дюймового дисковода на гибких дисках, показанный в листинге 11.5.

Листинг 11.5. Проверка наличия дисковода для гибких дисков

```
mov AH, 08h   ; функция 08h
mov DL, 00h   ; первый дисковод
int 13h       ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
cmp BL, 04h   ; проверяем тип дисковода
jne NO_144    ; нет такого дисковода по этому адресу
```

11.1.8. Функция *09h*

Функция позволяет инициализировать жесткий диск параметрами по умолчанию. Эта функция не используется для флоппи-дисководов.

Использование:

1. В регистр AH следует поместить код функции 09h.
2. В регистр DL следует поместить номер жесткого диска. Номера жестких дисков лежат в диапазоне 80h—FFh.
3. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен.

11.1.9. Функция *0Ch*

Данная функция предназначена для поиска указанного цилиндра и должна использоваться только для жестких дисков.

Использование:

1. В регистр AH следует поместить код функции 0Ch.
2. В регистр CH следует записать младшие 8 битов номера цилиндра. Номер цилиндра для жестких дисков имеет размер 10 бит. Два старших бита (8 и 9) определяются в регистре CL (биты 6 и 7).
3. В регистр CL записывается номер сектора (используются биты 0—5). В биты 6 и 7 следует записать значения старших битов для номера цилиндра.
4. В регистр DH нужно поместить номер головки.

5. В регистр DL следует поместить номер жесткого диска. Номера жестких дисков лежат в диапазоне 80h—FFh.
6. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. Например, чтобы переместиться на 110-й цилиндр (головка 1, сектор 1), следует использовать код из листинга 11.6.

Листинг 11.6. Поиск цилиндра на жестком диске

```
mov AH, 0Ch  ; функция 0Ch
mov CH, 6Eh  ; цилиндр номер 110
mov CL, 01h  ; первый сектор
mov DH, 01h  ; первая головка
mov DL, 80h  ; первый жесткий диск
int 13h      ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

11.1.10. Функция *0Dh*

Функция позволяет выполнить сброс жесткого диска. Происходит повторная инициализация контроллера, установленные параметры сбрасываются, а головки устанавливаются в позицию первой дорожки.

Использование:

1. В регистр AH следует поместить код функции 0Dh.
2. В регистр DL следует поместить номер жесткого диска. Номера жестких дисков лежат в диапазоне 80h—FFh.
3. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. Например, чтобы выполнить сброс второго жесткого диска, используйте следующий код из листинга 11.7.

Листинг 11.7. Сброс второго жесткого диска

```
mov AH, 0Ch  ; функция 0Dh
mov DL, 81h  ; второй жесткий диск
int 13h      ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

11.1.11. Функция *10h*

Эта функция позволяет проверить готовность жесткого диска к работе.

Использование:

1. В регистр AH следует поместить код функции 10h.
2. В регистр DL следует поместить номер жесткого диска. Номера жестких дисков лежат в диапазоне 80h—FFh.
3. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. Пример использования данной функции показан в листинге 11.8.

Листинг 11.8. Использование функции 10h

```
@repeat
mov AH, 10h  ; функция 10h
mov DL, 80h  ; первый жесткий диск
int 13h      ; вызываем прерывание
cmp AH, 0AAh ; если диск не готов
je  @repeat  ; повторяем
```

11.1.12. Функция *11h*

Данная функция позволяет выполнить рекалибровку жесткого диска. При этом контроллер переводит головки в область нулевого цилиндра.

Использование:

1. В регистр AH следует поместить код функции 11h.
2. В регистр DL следует поместить номер жесткого диска. Номера жестких дисков лежат в диапазоне 80h—FFh.
3. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1). Флаг CF в случае ошибки будет установлен в 1, иначе сброшен. Для устранения сбоев выполните рекалибровку второго диска, как показано в листинге 11.9.

Листинг 11.9. Рекалибровка жесткого диска

```
mov AH, 11h  ; функция 0Dh
mov DL, 81h  ; второй жесткий диск
int 13h      ; вызываем прерывание
jc  ErrorHnd ; если произошла ошибка, вызываем обработчик
```

11.1.13. Функция *14h*

Функция позволяет выполнить диагностику внутреннего контроллера жестких дисков.

Использование:

1. В регистр `AH` следует поместить код функции `14h`.
2. Вызвать прерывание `int 13h`.

Выход:

После выполнения функции в регистр `AH` будет записан код состояния (см. табл. 11.1), а в регистр `AL` — значение `00h`. Флаг `CF` в случае ошибки будет установлен в 1, иначе сброшен.

11.1.14. Функция *15h*

Эта функция позволяет получить тип установленного диска.

Использование:

1. В регистр `AH` следует поместить код функции `15h`.
2. В регистр `DL` следует поместить номер жесткого диска. Номера жестких дисков лежат в диапазоне `80h`—`FFh`, а номера гибких — `00h`—`7Fh`.
3. Вызвать прерывание `int 13h`.

Выход:

После успешного выполнения функции в регистр `AH` будет записан один из следующих кодов: `00h` — диск отсутствует, `01h` — флоппи-диск, `02h` — флоппи-диск или другое устройство со сменными носителями, `03h` — жесткий диск. Флаг `CF` в случае ошибки будет установлен в 1, иначе сброшен. В `CX:DX` будет помещено число секторов размером 512 байт. Не рекомендуется использовать данную функцию для проверки установленного типа диска.

11.1.15. Функция *16h*

Функция предназначена для отслеживания процесса смены носителя на флоппи-дисководе. Используйте данную функцию, если необходимо контролировать процесс замены дискеты в устройстве.

Использование:

1. В регистр AH следует поместить код функции 16h.
2. В регистр DL следует поместить номер флоппи-диска. Номера дисков лежат в диапазоне 00h—7Fh.
3. Вызвать прерывание int 13h.

Выход:

Если флаг CF сброшен, значит, изменения отсутствуют, и в регистр AH будет записано значение 00h (диск не менялся). Если флаг CF установлен в 1, значит, произошли изменения, и в регистр AH будет помещено определенное значение: 01h — недопустимая команда (для некоторых специфических устройств), 06h — диск был заменен или смена носителя не поддерживается, 80h — диск не готов или отсутствует носитель. Рассмотрим простой пример работы с данной функцией, представленный в листинге 11.10.

Листинг 11.10. Использование функции 16h

```
mov AH, 16h   ; функция 16h
mov DL, 00h   ; первый диск
int 13h       ; вызываем прерывание
jc  ErrorHnd ; произошла смена носителя, обрабатываем ситуацию
cmp AH, 06h   ; диск был заменен
je  DISK_CHANGE; передаем управление в процедуру обработки ситуации
cmp AH, 80h   ; диск не готов
je  NO_READY ; можем вызвать процедуру NO_READY для ожидания готовности
```

11.1.16. Функция *17h*

Данная функция устанавливает тип диска для последующей операции форматирования. Она используется для флоппи-дисков, но не поддерживает дискеты размером 1,44 Мб. Для работы с такими дискетами следует применять функцию 18h.

Использование:

1. В регистр AH следует поместить код функции 17h.
2. В регистр AL нужно записать код формата, имеющий одно из следующих значений: 01h — 320 или 360 Кб для диска размером 360 Кб, 02h — 320 или 360 Кб для диска размером 1,2 Мб, 03h — 1,2 Мб для диска размером 1,2 Мб, 04h — 720 Кб для диска размером 720 Кб.
3. В регистр DL следует поместить номер флоппи-диска. Номера дисков лежат в диапазоне 00h—7Fh.
4. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записан код состояния (см. табл. 11.1), а в регистр AL — значение 00h. Флаг CF в случае ошибки будет установлен в 1, иначе сброшен.

11.1.17. Функция *18h*

Функция позволяет установить тип носителя для последующей операции форматирования.

Использование:

1. В регистр AH следует поместить код функции 18h.
2. В регистр CH следует записать младшие 8 битов максимального количества цилиндров минус один.
3. В регистр CL нужно указать количество секторов на одной дорожке (биты 0—5), а также два старших бита (6 и 7) номера цилиндра.
4. В регистр DL следует поместить номер флоппи-диска. Номера дисков лежат в диапазоне 00h—7Fh.
5. Вызвать прерывание int 13h.

Выход:

После выполнения функции в регистр AH будет записано одно из следующих значений: 00h — требуемая комбинация значений поддерживается, 01h — ошибка выполнения функции, 0Ch — диск не поддерживается или имеет неизвестный тип, 80h — отсутствует носитель в дисководе. Кроме того, при успешном выполнении функции в ES:DI будет помещен указатель на таблицу параметров (обычно размером 11 байт).

* * *

На этом мы завершим рассмотрение стандартных функций и перейдем к описанию дополнительного набора функций, особенно актуального в современных операционных системах Windows. Эти функции используют прогрессивную логическую адресацию (LBA — Logical Block Address), что позволяет поддерживать устройства размером более 528 Мбайт (используя 28-битный адрес можно работать с дисками размером до 137,4 Гбайт). Кроме того, они позволяют работать с дисками в системах без поддержки BIOS и являются совместимыми с современными протоколами передачи данных (например, DMA). Для работы с дополнительными функциями также применяется прерывание int 13h. Список всех функций приведен в табл. 11.4. Прежде чем рассказать об этих функциях, я хотел бы объяснить, как нумеруются современные дисководы. Имеются два канала, каждый из которых поддерживает два устройства. Каналы нумеруются следующим образом: первый канал содержит устройства Device 0 (ведущее) и Device 1 (ведомое),

второй канал также содержит устройства Device 0 (ведущее) и Device 1 (ведомое). Нумерация дисков начинается с номера 80h и присваивается последовательно первому и второму устройствам на первом канале, а затем первому и второму устройствам на втором канале. Если жесткий диск подключен ведущим на первом канале, привод DVD подключен ведомым на этом же канале, а привод CD-RW подключен ведущим на втором канале, то нумерация устройств будет следующей: 80h — жесткий диск, 81h — привод DVD, 82h — привод CD-RW. Если, например, у вас установлены два устройства (жесткий диск и привод CD-RW) как ведущие на обоих каналах, нумерация будет выглядеть так: 80h — жесткий диск, 81h — привод CD-RW. Некоторые дополнительные функции (для записи, чтения и проверки данных) используют специальный пакет данных для указания адреса на диске. Сделано это для упрощения доступа к различным типам дисков, использующих различные способы адресации (CHS или LBA). Размер пакета должен быть не менее 16 байт и иметь формат, представленный в табл. 11.5.

Таблица 11.4. *Список дополнительных функций*

Код функции	Описание
41h	Проверка поддержки дополнительного набора функций
42h	Расширенное чтение данных
43h	Расширенная запись данных
44h	Проверка секторов
45h	Блокировка носителя
46h	Извлечение носителя
47h	Расширенный поиск
48h	Получение параметров устройства
49h	Получение расширенной информации о смене диска

Таблица 11.5. *Формат пакета данных для адреса*

Смещение	Размер	Описание
00h	BYTE	Размер пакета в байтах
01h	BYTE	Зарезервировано и равно 0
02h	BYTE	Количество блоков
03h	BYTE	Зарезервировано и равно 0
04h	DWORD	Адрес буфера хоста

Таблица 11.5 (окончание)

Смещение	Размер	Описание
08h	QWORD	Стартовый логический адрес
10h	QWORD	Линейный адрес буфера
18h	DWORD	Количество блоков
1Ch	DWORD	Зарезервировано и равно 0

Приведем краткий комментарий к табл. 11.5.

- Смещение 00h определяет размер пакета адреса в байтах. Это значение должно быть не меньше 16 байт, иначе вызываемая функция возвратит ошибку (AH равен 01h).
- Смещение 01h зарезервировано и должно быть установлено в 0.
- Смещение 02h определяет количество блоков с данными, которые должны быть переданы устройству. Максимальное значение не должно превышать 127 блоков, иначе вызываемая функция вернет ошибку (AH равен 01h). При установке байта в 0 никакие данные переданы не будут. Если установить значение байта в FFh, будет использоваться линейный адрес со смещением 10h, а количество блоков будет определяться значением в смещении 18h.
- Смещение 03h зарезервировано и должно быть установлено в 0.
- Смещение 04h определяет адрес буфера данных для хоста. Данный буфер будет использован для записи или чтения данных. Если значение адреса установлено в FFFFh:FFFFh, будет использован адрес буфера со смещением 10h.
- Смещение 08h определяет начальный логический адрес (64-разрядный) на диске, необходимый для передачи данных. Для устройств с CHS-адресацией заданное значение будет преобразовано, а для устройств с LBA-адресацией — передано без изменений. Пересчет LBA-адреса в CHS осуществляется по следующей формуле:

  $LBA = (C1 * H0 + H1) * S0 + S1 - 1$, где

  C1 — номер выбранного цилиндра;

  H0 — число головок (номер максимальной головки плюс один);

  H1 — номер выбранной головки;

  S0 — максимальное значение секторов;

  S1 — номер выбранного сектора.
- Смещение 10h определяет линейный адрес буфера данных для хоста. Данный буфер будет использован для записи или чтения данных. Это поле

используется, если в смещении 04h установлено значение FFFFh:FFFFh или количество блоков со смещением 02h установлено в FFh.

- Смещение 18h определяет количество блоков, которые должны быть переданы. Используется только в случае, если значение количества блоков в смещении 02h установлено в FFh.
- Смещение 1Ch зарезервировано и должно быть установлено в 0.

11.1.18. Функция *41h*

Функция позволяет проверить поддержку устройством дополнительного набора функций. Функция работает со всеми устройствами.

Использование:

1. В регистр AH следует поместить код функции 41h.
2. В регистр BX нужно записать значение 55AAh.
3. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
4. Вызвать прерывание int 13h.

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH будет записано значение 01h (недопустимая команда). Это значит, что дополнительный набор функций не поддерживается указанным устройством. Если флаг CF сброшен, результат выполнения будет распределен следующим образом: AH — номер версии дополнительного набора функций (для третьей версии значение равно 30h), AL — внутренняя информация BIOS, BX — код 55AAh, CX — битовая маска поддерживаемых возможностей (табл. 11.6). Установка какого-либо бита маски в 1 говорит о поддержке соответствующей возможности.

Таблица 11.6. *Битовая маска для функции 41h*

Бит	Описание
0	Поддерживаются функции доступа к жесткому диску (41h, 42h, 43h, 44h 47h и 48h)
1	Поддерживаются функции блокировки и смены носителя (45h, 46h, 48h и 49h)
2	Поддерживается функция 48h
3	Поддерживаются 64-битные расширения
4—15	Установлены в 0

В листинге 11.11 рассмотрим пример для определения поддержки дополнительного набора функций для второго устройства.

Листинг 11.11. Определение поддержки набора расширенных функций

```
mov AH, 41h  ; функция 41h
mov BX, 55AAh; обязательное значение
mov DL, 81h  ; второй диск
int 13h      ; вызываем прерывание
jc  NO_SUPPORT; набор дополнительных функций не поддерживается
cmp BX, 55AAh; проверяем полученное значение
jne ERROR_HND; произошла ошибка
test CX, 01b ; есть ли поддержка функций доступа
jz  NO_ACCESS; нет
test CX, 010b; есть ли поддержка функций блокировки и смены носителей
jz  NO_LOCK  ; нет
```

Этот пример чисто демонстрационный. В реальной программе следует распределить выполнение условий в зависимости от результата функции `41h`.

11.1.19. Функция *42h*

Эта функция позволяет прочитать данные с диска в выделенный буфер памяти.

Использование:

1. В регистр `AH` следует поместить код функции `42h`.
2. В регистр `DL` следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от `80h` до `83h` (четыре дисковода).
3. В `DS:SI` нужно записать пакет адреса для диска.
4. Вызвать прерывание `int 13h`.

Выход:

Флаг переноса `CF` в случае ошибки будет установлен в 1, а в регистр `AH` будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг `CF` будет сброшен, а в регистр `AH` записано значение `00h`. В случае ошибки поле счетчика индексов (в пакете адреса) будет содержать число реально прочитанных блоков данных. Рассмотрим пример кода для чтения первого сектора диска (листинг 11.12).

Листинг 11.12. Чтение первого сектора жесткого диска

```
; определяем буфер для возвращаемых данных размером 512 байт
DataSector DB 512 DUP (?)
```

```
; определяем массив для пакета адреса и обнуляем
DataPack DB 16 DUP (0)
; общий код программы
; . . .
; проверяем поддержку устройством дополнительных функций
mov AH, 41h  ; функция 41h
mov BX, 55AAh; обязательное значение
mov DL, 80h  ; первый диск
int 13h      ; вызываем прерывание
jc NO_SUPPORT; набор дополнительных функций не поддерживается
cmp BX, 55AAh; проверяем полученное значение
jne ERROR_HND; произошла ошибка
mov [DataPack], 16; указываем минимальный размер пакета адреса
mov [DataPack + 2], 1; нужно прочитать один блок
; указываем адрес буфера данных
mov [word ptr DataPack + 4], offset DataSector
mov AX, DS
mov [word ptr DataPack + 6], AX
; указываем логический адрес первого сектора на диске
mov [dword ptr DataPack + 8], 0
mov [dword ptr DataPack + 12], 0
mov AH, 42h  ; функция чтения данных
mov DL, 80h  ; первый диск
mov SI, offset DataPack; указатель на пакет адреса
int 13h      ; вызываем прерывание
```

11.1.20. Функция *43h*

Функция предназначена для записи данных на диск из указанного буфера памяти.

Использование:

1. В регистр `AH` следует поместить код функции `43h`.
2. В `AL` следует поместить условие записи (0 или 1 — для выполнения записи без проверки, 2 — с последующей проверкой записанных данных).
3. В регистр `DL` следует записать номер устройства. Нумерация устройств должна лежать в диапазоне от `80h` до `83h` (четыре дисковода).
4. В `DS:SI` нужно записать пакет адреса для диска.
5. Вызвать прерывание `int 13h`.

Выход:

Флаг переноса `CF` в случае ошибки будет установлен в 1, а в регистр `AH` будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг `CF` будет сброшен, а в регистр `AH` записано значение

00h. В случае ошибки поле счетчика индексов (в пакете адреса) будет содержать число реально записанных блоков данных. Рассмотрим демонстрационный пример записи первого сектора диска (листинг 11.13).

Листинг 11.13. Запись первого сектора жесткого диска

```
; определяем буфер для записываемых данных размером 512 байт
DataSector DB 512 DUP (0)
; определяем массив для пакета адреса и обнуляем
DataPack DB 16 DUP (0)
; общий код программы
; ...
; проверяем поддержку устройством дополнительных функций
mov AH, 41h  ; функция 41h
mov BX, 55AAh; обязательное значение
mov DL, 80h  ; первый диск
int 13h      ; вызываем прерывание
jc NO_SUPPORT; набор дополнительных функций не поддерживается
cmp BX, 55AAh; проверяем полученное значение
jne ERROR_HND; произошла ошибка
mov [DataPack], 16; указываем минимальный размер пакета адреса
mov [DataPack + 2], 1; нужно записать один блок
; указываем адрес буфера данных
mov [word ptr DataPack + 4], offset DataSector
mov AX, DS
mov [word ptr DataPack + 6], AX
; указываем логический адрес первого сектора на диске
mov [dword ptr DataPack + 8], 0
mov [dword ptr DataPack + 12], 0
mov AH, 43h  ; функция записи данных
mov DL, 80h  ; первый диск
mov SI, offset DataPack; указатель на пакет адреса
int 13h      ; вызываем прерывание
```

Не пытайтесь повторить этот код, чтобы не затереть загрузочную область диска! Пример приведен только для иллюстрации работы с функцией 43h.

11.1.21. Функция *44h*

Данная функция позволяет проверить указанные сектора без перемещения их между устройством и памятью.

Использование:

1. В регистр AH следует поместить код функции 44h.
2. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).

3. В DS:SI нужно записать пакет адреса для диска.
4. Вызвать прерывание int 13h.

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг CF будет сброшен, а в регистр AH будет записано значение 00h. В случае ошибки поле счетчика индексов (в пакете адреса) будет содержать число реально проверенных блоков данных.

11.1.22. Функция *45h*

Эта функция позволяет логически блокировать сменные носители для указанного устройства. Может применяться и для жестких дисков, если они поддерживают такую возможность (функция 41h). Каждой блокировке должна соответствовать обратная операция. После включения питания или перезагрузки системы все блокировки будут автоматически сняты.

Использование:

1. В регистр AH следует поместить код функции 45h.
2. В регистр AL следует указать тип операции: 00h — блокировать носитель в устройстве, 01h — разблокировать носитель в устройстве, 02h — получить состояние блокировки (есть или нет).
3. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
4. Вызвать прерывание int 13h.

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг CF будет сброшен, в регистр AH будет записано значение 00h, а в AL — состояние блокировки (1 — носитель блокирован, 0 — носитель разблокирован). Например, чтобы блокировать третий диск (например, привод CD-ROM), можно использовать код из листинга 11.14.

Листинг 11.14. Блокировка диска

```
mov AH, 45h  ; функция 45h
mov AL, 00h  ; блокировать носитель на диске
mov DL, 82h  ; номер диска
int 13h      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
mov AH, 45h  ; функция 45h
```

```
mov AL, 02h   ; проверяем состояние блокировки
mov DL, 82h   ; номер диска
int 13h       ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
cmp AL, 1     ; проверяем, блокировано ли устройство
jne NO_LOCK   ; устройство не заблокировано
```

11.1.23. Функция *46h*

Функция позволяет извлечь носитель из указанного дисковода, если последний поддерживает такую возможность.

Использование:

1. В регистр AH следует поместить код функции 46h.
2. В регистр AL следует записать значение 00h.
3. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
4. Вызвать прерывание int 13h.

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг CF будет сброшен, а в регистр AH будет записано значение 00h. Пример использования данной функции показан в листинге 11.15.

Листинг 11.15. Извлечение носителя из дисковода

```
mov AH, 41h   ; функция 41h
mov BX, 55AAh; обязательное значение
mov DL, 81h   ; второй диск
int 13h       ; вызываем прерывание
jc NO_SUPPORT; набор дополнительных функций не поддерживается
cmp BX, 55AAh; проверяем полученное значение
jne ERROR_HND; произошла ошибка
test CX, 010b; есть ли поддержка функций блокировки и смены носителей
jz  NO_LOCK   ; нет
mov AH, 46h   ; функция 46h
mov AL, 00h   ; всегда 0
mov DL, 81h   ; второй диск
int 13h       ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
```

11.1.24. Функция *47h*

Эта функция позволяет найти данные на диске, запрашиваемые компьютером (программой). При успешном выполнении функции считывающая механика диска будет установлена в заданную позицию.

Использование:

1. В регистр AH следует поместить код функции 47h.
2. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
3. В DS:SI нужно записать пакет адреса для диска.
4. Вызвать прерывание int 13h.

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг CF будет сброшен, а в регистр AH будет записано значение 00h. Функция возвратит управление, даже если операция поиска не будет закончена.

11.1.25. Функция *48h*

Функция позволяет получить различные параметры устройства. Она поддерживается всеми устройствами.

Использование:

1. В регистр AH следует поместить код функции 48h.
2. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
3. В DS:SI нужно записать адрес буфера данных.
4. Вызвать прерывание int 13h.

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг CF будет сброшен, а в регистр AH будет записано значение 00h. Буфер данных в DS:SI будет содержать запрашиваемую информацию. Формат буфера данных показан в табл. 11.7.

Таблица 11.7. *Формат буфера данных для функции* 48h

Байт	Тип	Описание
0—1	WORD	Размер выделенного буфера в байтах (должен быть не менее 30 байтов)

Таблица 11.7 *(окончание)*

Байт	Тип	Описание
2—3	WORD	Дополнительные информационные флаги
4—7	DWORD	Количество цилиндров по умолчанию
8—11	DWORD	Количество головок по умолчанию
12—15	DWORD	Количество секторов на одной дорожке по умолчанию
16—23	QWORD	Количество секторов
24—25	WORD	Размер одного сектора в байтах
26—29	DWORD	Указатель на расширенную таблицу параметров устройства (DPTE)
30—31	WORD	Значение 0BEDDh, идентифицирующее наличие пути для устройства
32	BYTE	Размер данных о пути к устройству (должен быть равен 44h)
33	BYTE	Резерв
34—35	WORD	Резерв
36—39	ASCII (4 байта)	Строковое значение для типа шины хоста
40—47	ASCII (8 байт)	Тип интерфейса
48—55	QWORD	Путь к интерфейсу
56—71	2 QWORD (16 байт)	Путь к устройству
72	BYTE	Резерв
73	BYTE	Контрольная сумма для данных, определяющих путь к устройству

Приведем краткое описание табл. 11.7.

- Байты 0—1 содержат размер в байтах выделяемого буфера, записываемый вызывающей программой. Если размер буфера меньше 30 байтов, функция не вернет информацию о параметрах устройства. Если размер буфера меньше 26 байтов, функция вернет ошибку.
- Байты 2—3 содержат 16-битное значение, отдельные биты которого определяют ту или иную возможность устройства. Формат этого значения представлен в табл. 11.8. Если бит установлен в 1, указанная возможность поддерживается.
- Байты 4—7 указывают на общее количество цилиндров в устройстве. Это значение будет на 1 больше максимального номера цилиндра. Отсчет ведется с 0.

***Таблица 11.8.** Формат информационного параметра устройства*

Бит	Описание
0	Поддержка сообщений об ошибках (нарушение границы) для DMA
1	Значения в байтах 4—15 являются истинными
2	Устройство поддерживает сменные носители
3	Устройство поддерживает проверку во время записи
4	Устройство поддерживает уведомляющие сообщения о смене носителя
5	Устройство поддерживает блокировку носителя
6	Выдаются максимальные значения для геометрии устройства, и носитель отсутствует
7	Поддерживается функция `50h` для доступа к устройству
8—15	Резерв

- Байты 8—11 указывают на общее количество головок в устройстве. Это значение будет на 1 больше максимального номера головки. Отсчет ведется с 0.
- Байты 12—15 указывают на общее количество секторов на одной дорожке. Это значение будет равно максимальному номеру сектора. Отсчет ведется с 1.
- Байты 16—23 определяют полное количество секторов на диске. Это значение будет на 1 больше максимального числа секторов. Отсчет ведется с 0.
- Байты 24—25 определяют количество байтов в одном секторе диска.
- Байты 26—29 определяют указатель на расширенную таблицу параметров устройства (DPTE — Device Parameter Table Extension). Данное поле действительно, если функция `41h` установила бит 2 регистра `CX` в 1. Если таблица параметров не поддерживается, это поле будет установлено в `FFFFh:FFFFh`. Кроме того, данное поле может присутствовать только в расширенном интерфейсе INT 13 и поддерживается как ATA-, так и ATAPI-устройствами.
- Байты 30—31 должны содержать значение `0BEDDh`, если присутствует информация о пути к устройству.
- Байт 32 определяет размер информационного блока о пути к диску. Должен быть установлен в `44h`.
- Байты 33, 34—35 зарезервированы и установлены в 0.
- Байты 36—39 определяют тип шины хоста, на которой установлено устройство. Они представляют собой четыре ASCII-символа и могут прини-

мать одно из перечисленных в табл. 11.9 значений. Данные должны быть выровнены и дополнены при необходимости символом пробела (`20h`).

Таблица 11.9. *Тип шины хоста*

Тип шины	Описание типа шины	Комбинация значений
PCI	PCI Local Bus	50h 43h 49h 20h
ISA	Legacy 16 bit fixed bus	49h 53h 41h 20h
PCI-X	PCI-X Bus	50h 43h 49h 58h
IBND	Infiniband	49h 42h 4Eh 44h
XPRS	PCI Express	58h 50h 52h 53h
HTPT	HyperTransport	48h 54h 50h 54h

- Байты 40—47 определяют тип интерфейса. Представляют собой восемь ASCII-символов и могут принимать одно из перечисленных в табл. 11.10 значений. Данные должны быть выровнены и дополнены при необходимости символом пробела (`20h`).

Таблица 11.10. *Тип интерфейса*

Тип	Описание интерфейса	Комбинация значений
ATA	ATA- и ATAPI-устройства, поддерживающие команды ATA	41h 54h 41h 20h 20h 20h 20h 20h
ATAPI	ATA- и ATAPI-устройства, поддерживающие команды ATAPI	41h 54h 41h 50h 49h 20h 20h 20h
SCSI	Устройства SCSI	53h 43h 53h 49h 20h 20h 20h 20h
USB	Устройства USB для хранения данных	55h 53h 42h 20h 20h 20h 20h 20h
1394	Устройства 1394 для хранения данных	31h 33h 39h 34h 20h 20h 20h 20h
FIBRE	Fibre Channel	46h 49h 42h 52h 45h 20h 20h 20h
I2O	Интеллектуальные устройства ввода-вывода	49h 32h 4Fh 20h 20h 20h 20h 20h
RAID	Дисковый массив для хранения данных	52h 41h 49h 44h 20h 20h 20h 20h
SATA	Serial ATA	53h 41h 54h 41h 20h 20h 20h 20h

- Байты 48—55 указывают на идентификатор пути к интерфейсу диска, который имеет размер 8 байт. Формат этого поля представлен в табл. 11.11.

Таблица 11.11. *Формат поля для идентификатора пути к интерфейсу диска*

Шина	Смещение	Размер	Описание
ISA	48	WORD	Базовый 16-битный адрес
	50	WORD	Зарезервировано и равно 0
	52	DWORD	Зарезервировано и равно 0
PCI	48	BYTE	Номер шины PCI (00h—FEh). Если значение равно FFh, значит, это поле не используется
	49	BYTE	Номер слота на шине PCI (00h—FEh). Если значение равно FFh, значит, это поле не используется
	50	BYTE	Номер функции PCI (00h—FEh). Если значение равно FFh, значит, это поле не используется
	51	BYTE	Номер канала. Позволяет различить первичный (00h) и вторичный каналы (01h). Возможные значения номера канала могут лежать в промежутке от 00h до FEh. Если значение равно FFh, значит, это поле не используется
	52	DWORD	Зарезервировано и равно 0
PCI-X	48	BYTE	Номер шины PCI-X (00h—FEh). Если значение равно FFh, значит, это поле не используется
	49	BYTE	Номер слота на шине PCI-X (00h—FEh). Если значение равно FFh, значит, это поле не используется
	50	BYTE	Номер функции PCI-X (00h—FEh). Если значение равно FFh, значит, это поле не используется
	51	BYTE	Номер канала. Позволяет различить первичный (00h) и вторичный каналы (01h). Возможные значения номера канала могут лежать в промежутке от 00h до FEh. Если значение равно FFh, значит, это поле не используется
	52	DWORD	Зарезервировано и равно 0
IBND	48h	QWORD	Не определено
XPRS	48h	QWORD	Не определено
HTPT	48h	QWORD	Не определено

- Байты 56—71 определяют идентификатор пути для устройства согласно табл. 11.12.

Таблица 11.12. Формат поля для идентификатора пути к диску

Тип	Смещение	Размер	Описание
ATA	56	BYTE	Номер ATA-устройства (00h — первое Device 0, 01h — второе Device 1)
	57	BYTE	Зарезервировано и равно 0
	58	WORD	Зарезервировано и равно 0
	60	DWORD	Зарезервировано и равно 0
	64	QWORD	Зарезервировано и равно 0
ATAPI	56	BYTE	Номер ATAPI-устройства (00h — первое Device 0, 01h — второе Device 1)
	57	BYTE	Логический номер устройства
	58	BYTE	Зарезервировано и равно 0
	59	BYTE	Зарезервировано и равно 0
	60	DWORD	Зарезервировано и равно 0
	64	QWORD	Зарезервировано и равно 0
SCSI	56	WORD	Идентификатор физического устройства SCSI
	58	QWORD	Логический номер устройства
	66	WORD	Зарезервировано и равно 0
	68	DWORD	Зарезервировано и равно 0
USB	56	QWORD	Номер устройства размером 64 бита
	64	QWORD	Зарезервировано и равно 0
1394	56	QWORD	Уникальный идентификатор устройства (EUI-64) размером 64 бита
	64	QWORD	Зарезервировано и равно 0
FIBRE	56	QWORD	Общемировой идентификатор устройства (WWID) размером 64 бита
	64	QWORD	Логический номер устройства
I2O	56	QWORD	Идентификатор устройства размером 64 бита
	64	QWORD	Зарезервировано и равно 0
RAID	56	DWORD	Номер RAID-массива, в который входит данное устройство
	60	DWORD	Зарезервировано и равно 0
	64	QWORD	Зарезервировано и равно 0

Таблица 11.12 (окончание)

Тип	Смещение	Размер	Описание
SATA	56	BYTE	Номер SATA-устройства (00h — первое Device 0, 01h — второе Device 1)
	57	BYTE	Зарезервировано и равно 0
	58	WORD	Зарезервировано и равно 0
	60	DWORD	Зарезервировано и равно 0
	64	QWORD	Зарезервировано и равно 0

- Байт 72 зарезервирован.
- Байт 73 определяет значение контрольной суммы для данных пути к устройству с учетом сигнатуры 0BEDDh. Вычисляется в виде двоичного дополнения суммы значений без знака для смещений от 30 до 72. Сумма значений для диапазона 30—73 равна 0.

Для приложений, не использующих расширенный интерфейс INT 13, можно воспользоваться расширенной таблицей текущих параметров устройства (DPTE). Формат таблицы показан в табл. 11.13.

***Таблица 11.13.** Формат расширенной таблицы параметров диска*

Смещение	Размер	Описание
0—1	WORD	Базовый адрес порта ввода-вывода
2—3	WORD	Адрес управляющего регистра
4	BYTE	Регистр устройства
5	BYTE	Зарезервирован для BIOS
6	BYTE	Номер прерывания
7	BYTE	Число повторений для многоразовых команд чтения и записи
8	BYTE	Информация о DMA
9	BYTE	Тип пакетного режима
10—11	WORD	Дополнительная информация
12—13	WORD	Зарезервировано и равно 0
14	BYTE	Определяет версию таблицы и равно 11h (в третьей версии)
15	BYTE	Контрольная сумма

Приведем краткое пояснение к табл. 11.3.

- Байты 0—1 указывают на 16-разрядный базовый адрес порта устройства для операций ввода-вывода.
- Байты 2—3 указывают на 16-разрядный адрес управляющего регистра устройства.
- Байт 4 содержит текущий формат управляющего регистра устройства (табл. 11.14).

Таблица 11.14. *Формат управляющего регистра устройства*

Бит	Описание
0—3	Равны 0
4	Бит номера устройства (0 — Device 0, 1 — Device 1)
5	Равен 1
6	Режим LBA (1 — используется)
7	Равен 1

- Байт 5 используется BIOS для внутренних целей.
- Байт 6 — номер выделенного устройству прерывания. Биты 4—7 всегда равны 0, а биты 0—3 определяют номер прерывания.
- Байт 7 определяет количество повторений для передачи блоков данных, используемых в некоторых специфических командах ATA (команды для многократной записи или чтения).
- Байт 8 указывает на номер канала DMA (биты 0—3) и тип режима DMA (4—7).
- Байт 9 определяет параметры режима PIO (Programmed Input/Output — программируемый ввод-вывод). Биты 4—7 равны 0, а биты 0—3 содержат тип режима PIO.
- Байты 10—11 определяют дополнительные флаги. Формат значения этого поля показан в табл. 11.15. Если бит установлен в 1, опция используется.

Таблица 11.15. *Дополнительные флаги*

Бит	Описание
0	Используется режим Fast PIO. Если этот бит установлен, информация в байте 9 доступна
1	Используется DMA. Если этот бит установлен, информация в байте 8 доступна

Таблица 11.15 (окончание)

Бит	Описание
2	Используются команды многократной записи или чтения
3	Используется адресация для эмуляции CHS
4	Используется адресация LBA
5	Устройство поддерживает сменные носители (например, CD-ROM)
6	Устройство поддерживает пакетные команды
7	Устройство поддерживает 32-разрядный режим передачи данных
8	Определяет готовность устройства ATAPI к передаче данных, если бит 6 равен 1
9—10	Определяет тип преобразования адресов для CHS (`00h` — поразрядный сдвиг, `01h` — LBA, `02h` — резерв и `03h` — определяется производителем), если бит 3 равен 1
11	Используется режим Ultra DMA совместно с байтом 8
12—15	Зарезервировано и равно 0

- Байты 12—13 зарезервированы и должны быть равны 0.
- Байт 14 определяет версию таблицы и должен быть равен `11h`.
- Байт 15 определяет контрольную сумму для всех байтов в таблице.

Рассмотрим простой пример для получения информации о параметрах устройства (листинг 11.16).

Листинг 11.16. Получение информации об устройстве

```
; определяем буфер для возвращаемых данных
DiskInfo DB 74 DUP (?)
; общий код программы
; ...
; проверяем поддержку устройством дополнительных функций
mov AH, 41h  ; функция 41h
mov BX, 55AAh; обязательное значение
mov DL, 81h  ; второй диск
int 13h      ; вызываем прерывание
jc NO_SUPPORT; набор дополнительных функций не поддерживается
cmp BX, 55AAh; проверяем полученное значение
jne ERROR_HND; произошла ошибка
; получаем информацию для указанного устройства
mov AH, 48h  ; функция 48h
```

```
mov DL, 81h  ; второй диск
mov [word ptr DiskInfo], 30; указываем минимальный размер блока данных
mov SI, offset DiskInfo; передаем указатель на буфер данных
int 13h      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
```

11.1.26. Функция *49h*

Функция позволяет определить, была ли выполнена пользователем смена носителя. При каждой попытке заменить носитель устройство вырабатывает определенный сигнал.

Использование:

1. В регистр `AH` следует поместить код функции `49h`.
2. В регистр `DL` следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от `80h` до `83h` (четыре дисковода).
3. Вызвать прерывание `int 13h`.

Выход:

Флаг переноса `CF` будет установлен в 1 (`AH` равен `06h`), если произошла смена носителя, и сброшен в обратном случае (`AH` равен `00h`). В листинге 11.17 проверяется момент замены (или попытки замены) носителя.

Листинг 11.17. Проверка сигнала смены носителя

```
@repeat:
mov AH, 49h  ; функция 49h
mov DL, 80h  ; первый диск
int 13h      ; вызываем прерывание
jnc @repeat  ; повторяем опрос, если замена не производилась
```

11.1.27. Функция *4Eh*

Эта функция позволяет установить конфигурацию аппаратного обеспечения для работы с дисками. С помощью данной функции можно сконфигурировать контроллер на материнской плате так, чтобы максимально оптимизировать работу указанного устройства. Несмотря на то, что один канал поддерживает два дисковода, эта функция выполнит конфигурацию только для одного.

Использование:

1. В регистр `AH` следует поместить код функции `4Eh`.
2. В регистр `AL` нужно записать номер команды (табл. 11.16).

3. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
4. Вызвать прерывание int 13h.

Таблица 11.16. Список команд

Код команды	Описание
0h	Использовать предвыборку
1h	Не использовать предвыборку
2h	Использовать максимально быстрый режим PIO
3h	Использовать режим PIO 0
4h	Восстановить режим PIO, используемый по умолчанию
5h	Использовать максимально быстрый режим DMA
6h	Блокировать доступ к DMA для прерывания int 13h

Выход:

Флаг переноса CF в случае ошибки будет установлен в 1, а в регистр AH будет записано значение кода ошибки (см. табл. 11.1). При успешном выполнении операции флаг CF будет сброшен, в регистр AH будет записано значение 00h, а в регистр AL — код выполнения (0 — команда выполнена правильно, 1 — выполнение команды повлияло на другие устройства).

Рассмотрим небольшой пример для установки максимально быстрого режима PIO, представленный в листинге 11.18.

Листинг 11.18. Установка наилучшего режима PIO

```
mov AH, 4Eh   ; функция 4Eh
mov AL, 2h    ; самый быстрый режим PIO
mov DL, 82h   ; третий диск
int 13h       ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
cmp AL, 0     ; проверяем корректность выполнения функции
jnz ERROR_HND; некорректное выполнение функции
```

11.1.28. Функция *50h*

Функция позволяет подготовить BIOS к передаче данных для устройства, использующего пакетный режим работы (например, привод CD-ROM).

Использование:

1. В регистр AH следует поместить код функции 50h.
2. В регистр AL нужно записать номер подфункции D7h.
3. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
4. В ES:SI следует записать указатель на форматированную пакетную команду согласно табл. 11.17.
5. Вызвать прерывание int 13h.

Таблица 11.17. Формат пакетной команды

Смещение	Размер	Описание
0	WORD	Размер пакета в байтах
2—n	BYTE	Отформатированные данные пакетной команды

Выход:

При успешном выполнении флаг CF будет очищен, а в регистр AH будет записано значение 00h. Если произошла ошибка, флаг CF будет установлен, а в регистр AH будет записано одно из следующих значений ошибки: 01h — функция не поддерживается, 80h — команда не была завершена, 97h — подфункция D7h не поддерживается данным устройством, C3h — отформатированный блок пакетной команды имеет размер менее допустимого.

Данная команда помогает использовать подмножество команд SCSI (Small Computer System Interface — системный интерфейс малых вычислительных машин) для управления устройствами, работающими в пакетном режиме: ATAPI, 1394, USB и SCSI. Что такое пакетные команды и для чего они нужны, будет рассматриваться далее в этой главе. В табл. 11.18 показан стандартный формат пакетной команды, указатель на который помещается в регистр ES:SI.

Таблица 11.18. Формат 16-байтной пакетной команды

Смещение	Размер	Описание
0—1	WORD	Размер пакета команды в байтах
2	BYTE	Дополнительная информация
3	BYTE	Код пакетной команды
4—7	DWORD	Указатель на пакет данных команды
8—15	DWORD	Количество байт для передачи следующего блока данных

Таблица 11.18 (окончание)

Смещение	Размер	Описание
16—19	DWORD	Указатель на начало буфера данных
20—21	WORD	Время ожидания
22—23	WORD	Приращение времени

Приведем краткое описание таблицы.

- Байты 0—1 определяют размер пакетной команды в байтах.
- Байт 2 содержит дополнительную информацию. Биты 0—5 зарезервированы и должны быть установлены в 0. Биты 6 и 7 определяют направления передачи данных:
  - 00h — передается только команда без дополнительных данных;
  - 01h — данные передаются от устройства к компьютеру;
  - 02h — данные передаются от компьютера к устройству;
  - 03h — будет выполнен сброс интерфейса.
- Байт 3 содержит код пакетной команды или 0, если команда не используется.
- Байты 4—7 должны содержать указатель на дополнительные данные пакетной команды, которые будут переданы. Если данное поле установлено в ноль, оно игнорируется.
- Байты 8—15 определяют число байтов, которые будут переданы со следующей командой.
- Байты 16—19 определяют начало буфера данных, которые должны быть переданы.
- Байты 20—21 содержат время ожидания в миллисекундах перед началом посылки пакетной команды. Если это поле установлено в 0, функция вернет ошибку (80h).
- Байты 22—23 определяют приращение времени ожидания. Количество передаваемых данных делится на размер сектора и умножается на заданное в этом поле значение. Полученный результат будет добавлен к общему времени ожидания. Максимальное значение не должно превышать FFFFh. Если выполняется передача только кода команды или сброс интерфейса, данное поле игнорируется.

И еще разберем одну функцию, использующую прерывание int 15h и имеющую отношение к интерфейсу INT 13.

11.1.29. Функция *52h*

Функция позволяет извлечь из устройства сменный носитель. Она вызывается BIOS при использовании команды 46h. Программы кэширования могут перехватывать данное прерывание для своевременного сохранения данных на диске.

Использование:

1. В регистр AH следует поместить код функции 52h.
2. В регистр DL следует поместить номер устройства. Нумерация устройств должна лежать в диапазоне от 80h до 83h (четыре дисковода).
3. Вызвать прерывание int 15h.

Выход:

При успешном выполнении флаг CF будет очищен, а в регистр AH будет записано значение 00h. Если произошла ошибка, флаг CF будет установлен, а в регистр AH будет записано одно из следующих значений ошибки: B1h — носитель блокирован, B3h — носитель недоступен. Рассмотрим пример кода для извлечения носителя из устройства CD-ROM (установлен ведущим на втором канале), показанный в листинге 11.19.

Листинг 11.19. Извлечение сменного носителя из устройства CD-ROM

```
; предположим, что имеется только два дисковода: жесткий и CD-ROM
mov AH, 52h  ; функция 52h
mov DL, 81h  ; второй диск
int 15h      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
```

А теперь дополнительно рассмотрим набор функций BIOS для поддержки загрузочных дисков. Для работы с этими функциями используются две стандартные структуры данных: для описания технических требований (табл. 11.19) и типа передаваемой команды (табл. 11.21).

***Таблица 11.19.** Формат структуры описания технических требований*

Смещение	Размер	Описание
0	BYTE	Размер пакета (13h)
1	BYTE	Тип носителя
2	BYTE	Номер диска
3	BYTE	Индекс контроллера CD-диска
4—7	DWORD	Логический адрес

Таблица 11.19 (окончание)

Смещение	Размер	Описание
8—9	WORD	Технические требования для загрузочного диска
10—11	WORD	Размер сегмента данных пользователя
12—13	WORD	Сегмент загрузки для диска
14—15	WORD	Счетчик количества секторов
16	BYTE	Младшее значение числа цилиндров
17	BYTE	Количество секторов и старшее значение числа цилиндров
18	BYTE	Число головок

Приведем краткий комментарий к таблице.

- Байт 0 определяет размер всего пакета и должен быть установлен в 13h.
- Байт 1 определяет тип носителя и дополнительные данные. Формат этого байта приведен в табл. 11.20.

***Таблица 11.20.** Формат байта описания типа носителя*

Бит	Описание
0—3	Тип эмулируемого устройства (00h — эмуляция отсутствует, 01h — диск размером 1,2 Мб, 02h — диск размером 1,44 Мб, 03h — диск размером 2,88 Мб, 04h — жесткий диск C:)
4—5	Зарезервированы и должны иметь значение 0
6	Установка бита в 1 позволяет использовать интерфейс ATAPI (биты 8 и 9 должны быть установлены для устройств IDE)
7	Установка бита в 1 позволяет использовать интерфейс SCSI (биты 8 и 9 должны быть установлены для устройств SCSI)

- Байт 2 определяет номер диска. Возможны следующие значения: 00h — для эмуляции образа гибкого диска, 80h — для автозагрузочного жесткого диска, 81h—FFh — без автозагрузки и эмуляции.
- Байт 3 должен указывать на номер контроллера для устройства компакт-дисков.
- Байты 4—7 определяют логический адрес для диска.
- Байты 8—9 определяют параметры адресации дисков для интерфейсов SCSI и IDE. Если используется SCSI, байт 8 указывает на логический номер устройства (LUN), а байт 9 — на номер шины. Если используется

интерфейс IDE, байт 8 указывает на номер устройства (0 для ведущего или 1 для ведомого), а байт 9 не применяется.

- Байты 10—11 определяют размер сегмента данных для пользователя.
- Байты 12—13 определяют загрузочный сегмент, используемый функцией 4Ch. Если записать в это поле значение 00h, будет использован стандартный сегмент 7C0h.
- Байты 14—15 определяют значение виртуального счетчика секторов, который используется в функции 4Ch.
- Байт 16 должен содержать младшие 7 бит для количества цилиндров. Данное значение возвращает функция 08h в регистре CL.
- Байт 17 состоит из двух частей. Биты 0—5 должны определять количество секторов, а биты 6 и 7 — описывать старшие два разряда для количества цилиндров.
- Байт 18 определяет количество головок диска. Данное значение возвращает функция 08h в регистре DH.

Таблица 11.21. *Формат структуры для типа передаваемой команды*

Смещение	Размер	Описание
0	BYTE	Размер структуры (08h)
1	BYTE	Количество секторов в каталоге начальной загрузки, которые требуется передать
2	DWORD	Указатель на адрес буфера, который используется для каталога начальной загрузки
6	WORD	Значение первого сектора в каталоге начальной загрузки (00h)

11.1.30. Функция *4Ah*

Данная функция позволяет инициализировать эмуляцию диска.

Использование:

1. В регистр AH следует поместить код функции 4Ah.
2. В регистр AL нужно записать значение 00h.
3. В DS:SI необходимо записать указатель на пакет описания технических требований.
4. Вызвать прерывание int 13h.

Выход:

При успешной инициализации эмуляции диска флаг CF будет сброшен, а если система не поддерживает режим эмуляции — установлен. В регистр AH будет записан код выполнения (см. табл. 11.1).

11.1.31. Функция *4Bh*

Эта функция позволяет завершить режим эмуляции диска и восстанавливает используемое по умолчанию загрузочное устройство. Кроме того, данную функцию можно использовать для проверки установленного режима эмуляции.

Использование:

1. В регистр AH следует поместить код функции 4Bh.
2. В регистр AL нужно записать код команды. Поддерживаются следующие значения: 00h — возвратить состояние и завершить режим эмуляции, 01h — возвратить только текущее состояние.
3. В регистр DL следует записать номер диска, для которого будет завершен режим эмуляции. Если установить значение 7Fh, режим эмуляции будет завершен для всех дисков.
4. В DS:SI необходимо записать указатель на пустой пакет описания технических требований.
5. Вызвать прерывание int 13h.

Выход:

При успешном завершении режима эмуляции диска флаг CF будет сброшен, а если система не находится в режиме эмуляции — установлен. В регистр AH будет записан код выполнения (см. табл. 11.1). В DS:SI будет записан пакет описания технических требований.

11.1.32. Функция *4Ch*

Функция позволяет инициализировать эмуляцию диска и выполнить перезагрузку системы.

Использование:

1. В регистр AH следует поместить код функции 4Ch.
2. В регистр AL нужно записать значение 00h.
3. В DS:SI необходимо записать указатель на пакет описания технических требований.
4. Вызвать прерывание int 13h.

Выход:

Если функция выполнена успешно, возвращаемое значение отсутствует.

* * *

На этом мы завершим описание функций BIOS и перейдем к непосредственному программированию портов.

11.2. Использование портов

Работа напрямую с портами является не такой сложной задачей, как может показаться сразу. Однако для правильной работы требуется хорошо знать структуру регистров дисковых устройств, а также иметь представление о протоколах работы с различными типами команд и интерфейсов. Сначала мы познакомимся с программированием накопителей на гибких дисках (флоппи-дисководов).

11.2.1. Регистры флоппи-дисковода

Для работы с данным устройством используются порты `3F0h`—`3F7h`. Каждый из этих портов взаимодействует с определенным регистром дисковода. Назначение каждого регистра приведено в табл. 11.22. Некоторые из них выполняют двойную роль.

Таблица 11.22. *Список портов ввода-вывода для флоппи-дисковода*

Регистр	Режим работы	Описание
`3F0h`	Чтение	Зарезервирован
`3F1h`	Чтение и запись	Дополнительный регистр состояния (SRB)
`3F2h`	Чтение и запись	Регистр цифрового вывода (DOR)
`3F3h`	Чтение и запись	Регистр управления лентопротяжным механизмом (TDR)
`3F4h`	Чтение	Основной регистр состояния (MSR)
`3F4h`	Запись	Регистр управления скоростью передачи данных (DSR)
`3F5h`	Чтение и запись	Регистр данных (FIFO)
`3F6h`	—	Зарезервирован
`3F7h`	Чтение	Регистр цифрового ввода (DIR)
`3F7h`	Запись	Регистр управления конфигурацией (CCR)

Рассмотрим каждый из регистров подробнее.

11.2.1.1. Дополнительный регистр состояния

Этот регистр определяет текущее состояние устройства. Может применяться как для чтения, так и для записи. Использует порт под номером `3F1h`. Формат регистра показан в табл. 11.23.

Таблица 11.23. *Формат дополнительного регистра состояния*

Биты	7	6	5	4	3	2	1	0
Описание	Резерв					УП	Резерв	СП
Чтение	Резерв					0	Резерв	Простой
Запись	Резерв					Простой	Резерв	Резерв

Приведем краткий комментарий к таблице.

- Бит 0 определяет текущее состояние простоя (СП) устройства и доступен в режиме чтения регистра.
- Бит 2 служит для управления режимом простоя (УП). Если бит установлен в 1, то блокируется возможность выключения питания через регистр DSR (Datarate Select Register). Восстановить нормальный режим можно только с помощью перезагрузки системы.

Для использования этого регистра необходимо, чтобы он был включен управляющей командой (бит EREG EN установлен в 1). Бит EREG EN (Enhanced Register Enable) применяется для включения дополнительного регистра статуса. Пример считывания данного регистра приведен в листинге 11.20.

Листинг 11.20. Считывание информации из регистра состояния

```
xor AL, AL   ; обнуляем регистр
mov DX, 3F1h ; указываем порт
in  AL, DX   ; читаем из порта один байт
test AL, 01b ; проверяем бит 0
je  POWER_OFF; питание отключено
jne POWER_ON ; питание включено
```

Аналогичный пример на C++ показан в листинге 11.21.

Листинг 11.21. Считывание информации из регистра состояния в C++

```
// пишем функцию для проверки состояния питания
bool InPowerDown ()
{
   DWORD dwResult  = 0; // переменная для хранения результата
```

```
    // читаем состояние порта
    inPort ( 0x3F1, &dwResult, 1);
    if ( ( dwResult & 0x01) == 0x01) return true;
    return false;
}
```

11.2.1.2. Регистр цифрового вывода

Этот регистр позволяет управлять мотором флоппи-дисковода. Он использует порт под номером `3F2h`. Может применяться как для чтения, так и для записи. Формат регистра показан в табл. 11.24.

Таблица 11.24. *Формат регистра цифрового вывода*

Биты	7	6	5	4	3	2	1	0
Описание	Резерв	Резерв	Мотор	Мотор	DMA	Сброс	Резерв	Диск

Приведем комментарий к таблице.

- Бит 0 позволяет выбрать номер устройства. Возможны следующие значения: 0 — первый дисковод (`1Ch`) или 1 — второй дисковод (`2Dh`).
- Бит 1 зарезервирован и должен быть установлен в 0.
- Бит 2, установленный в 0, позволяет выполнить сброс устройства. После того как выполнена операция сброса, данный бит устанавливается в 1.
- Бит 3, установленный в 0, блокирует прерывания и доступ устройства к DMA, иначе доступ разрешен.
- Бит 4 управляет двигателем для первого (Drive 0) дисковода (1 — двигатель запущен, 0 — двигатель выключен).
- Бит 5 управляет двигателем для второго (Drive 1) дисковода (1 — двигатель запущен, 0 — двигатель выключен).
- Биты 6 и 7 зарезервированы и должны быть установлены в 0.

Перед началом работы с дисководом следует включить двигатель. Для запуска двигателя первого дисковода можно применить код из листинга 11.22.

Листинг 11.22. Включение двигателя флоппи-дисковода

```
mov DX, 3F2h ; указываем порт
mov AL, 0    ; выполняем сброс
out DX, AL   ; записываем значение в порт
mov AL, 1Ch  ; инициализируем первый дисковод и запускаем двигатель
out DL, AL   ; записываем значение в порт
mov CX, 4000 ; устанавливаем небольшое ожидание для разгона двигателя
@delay:
loop @delay  ; двигатель раскрутится после завершения цикла
```

Для выключения двигателя достаточно выполнить код из листинга 11.23.

Листинг 11.23. Выключение двигателя флоппи-дисковода

```
mov DX, 3F2h ; указываем порт
mov AL, 0Ch  ; останавливаем двигатель
out DL, AL   ; записываем значение в порт
```

Аналогичный пример на C++ показан в листинге 11.24.

Листинг 11.24. Управление двигателем флоппи-дисковода в C++

```
// пишем функцию для инициализации дисковода и управления двигателем
void InitFDD ( bool Drive)
{
   if ( Drive) // выполнить инициализацию и включить двигатель
   {
     outPort ( 0x3F2, 0x1C, 1);
     Sleep ( 500); // ожидаем 0,5 секунды, пока двигатель раскрутится
   }
   else // выключить двигатель
   {
     outPort ( 0x3F2, 0x0C, 1);
   }
}
```

11.2.1.3. Регистр управления лентопротяжным механизмом

Данный регистр позволяет выбирать для определенного диска используемый лентопротяжный механизм. После этого любые обращения к диску будут переданы выбранному механизму. Восстановить регистры можно только через аппаратный сброс. Он использует порт под номером `3F3h`. Может применяться как для чтения, так и для записи. Формат регистра показан в табл. 11.25. Описание данного регистра рассматривается для контроллера фирмы Intel.

Таблица 11.25. *Формат регистра управления лентопротяжным механизмом*

Биты	7	6	5	4	3	2	1	0
Описание	Резерв					Авто	ЛМ 1	ЛМ 0
Чтение						Авто	ЛМ 1	ЛМ 0
Запись	0	0	0	0	0	Авто	ЛМ 1	ЛМ 0

Приведем краткое описание таблицы.

- Бит 0, установленный в 1, позволяет блокировать первый лентопротяжный механизм для первого дисковода (Drive 0). По умолчанию первый дисковод является загрузочным.
- Бит 1, установленный в 1, позволяет блокировать второй лентопротяжный механизм для первого дисковода (Drive 0).
- Бит 3 управляет выбором загрузочного дисковода. По умолчанию бит равен 0. При этом первому дисководу соответствует первый лентопротяжный механизм, а второму дисководу — второй. Если бит равен 1, первому дисководу будет сопоставляться второй лентопротяжный механизм, а второму — первый.

Пользоваться этим регистром не рекомендуется из-за различий использования его производителями (Intel, Via).

11.2.1.4. Основной регистр состояния

Этот регистр применяется для получения информации о завершении различных управляющих команд. Использует порт под номером `3F4h`. Может применяться только для чтения. Формат регистра показан в табл. 11.26.

Таблица 11.26. *Формат основного регистра состояния*

Биты	7	6	5	4	3	2	1	0
Описание	Данные	НП	Не DMA	Ком.	Резерв	Резерв	Диск 1	Диск 0

Приведем краткий комментарий к таблице.

- Бит 0 установлен в 1, когда первый дисковод (Drive 0) ищет данные команды или находится в режиме калибровки.
- Бит 1 установлен в 1, когда второй дисковод (Drive 1) ищет данные команды или находится в режиме калибровки.
- Биты 2 и 3 зарезервированы и равны 0.
- Бит 4 установлен в 1, когда происходит выполнение команды.
- Бит 5, установленный в 1, определяет передачу данных без использования режима DMA.
- Бит 6 определяет направление передачи (НП) данных. Если бит равен 0, передача осуществляется от хоста к флоппи-дисководу, а когда бит установлен в 1 — от флоппи-дисковода к хосту.
- Бит 7 информирует о готовности дисковода к приему данных. Если бит равен 1, можно передавать данные, если же бит установлен в 0, дисковод не готов к приему данных и следует подождать.

Например, если регистр содержит значение 90h, значит, выполняется команда и требуется подождать. А если в регистре записано значение 80h, значит, контроллер готов принять очередную команду.

11.2.1.5. Регистр управления скоростью передачи данных

Регистр позволяет настроить параметры для скорости передачи данных. Управление скоростью передачи необходимо для совместимости с более старыми устройствами. Использует порт под номером 3F4h. Может применяться только для записи. Формат регистра показан в табл. 11.27.

Таблица 11.27. *Формат регистра управления скоростью передачи*

Биты	7	6	5	4	3	2	1	0
Описание	Сброс	П	ЗКГ	Предкомпенсация			Скорость	

Приведем краткое описание таблицы.

- Биты 0—1 позволяют установить одну из четырех возможных скоростей передачи данных. Список поддерживаемых значений показан в табл. 11.28. По умолчанию используется скорость 250 Кбит в секунду.

Таблица 11.28. *Список скоростей*

Значение битов 0 и 1	Скорость
00b	500 Кбит/сек
01b	300 Кбит/сек
10b	250 Кбит/сек
11b	1 Мбит/сек

- Биты 2—4 определяют время задержки предкомпенсации. Это необходимо, поскольку магнитные носители имеют физически обоснованные задержки при намагничивании. Для компенсации магнитных явлений используется задержка по времени, измеряемая в наносекундах. Возможные значения для этого поля представлены в табл. 11.29. По умолчанию используются следующие задержки: для скорости передачи 1 Мбит/сек — 41,67 нс, а для скоростей 250—500 Кбит/сек — 125 нс.
- Бит 5 управляет питанием задающего кварцевого генератора (ЗКГ). Установка бита в 1 позволит отключить подачу питания на генератор. Изменение этого бита следует выполнять только в режиме пониженного энергопотребления.

Таблица 11.29. *Значения времени задержки*

Код задержки	Время, нс
000b	По умолчанию
001b	41,67
010b	83,34
011b	125,00
100b	166,67
101b	208,33
110b	250,00
111b	0,00 (не используется)

- Бит 6 управляет подачей питания к дисководу. При установке бита в 1 флоппи-дисковод будет переведен в режим пониженного потребления энергии. Аппаратный или программный сброс восстанавливает полный уровень подачи питания.
- Бит 7, установленный в 1, позволяет выполнить программный сброс контроллера, после чего он устанавливается в 0.

Рассмотрим пример кода для установки скорости передачи данных, показанный в листинге 11.25.

Листинг 11.25. Установка желаемой скорости

```
mov DX, 3F2h ; указываем порт
mov AL, 0    ; выполняем сброс
out DX, AL   ; записываем значение в порт
mov AL, 1Ch  ; инициализируем первый дисковод и запускаем двигатель
out DL, AL   ; записываем значение в порт
mov CX, 4000 ; устанавливаем небольшое ожидание для разгона двигателя
mov DX, 3F4h ; указываем порт
mov AL, 13   ; 300 Кбит/сек и 125,00 нс
out DX, AL   ; записываем значение в порт
```

Аналогичный пример для C++ представлен в листинге 11.26.

Листинг 11.26. Установка желаемой скорости в C++

```
// раскручиваем двигатель и выполняем инициализацию
outPort ( 0x3F2, 0x1C, 1);
```

```
// ожидаем 0,5 секунды, пока двигатель раскрутится
Sleep ( 500);
// устанавливаем скорость 300 Кбит/сек и время 125,00 нс
outPort ( 0x3F4, 0x0D, 1);
```

11.2.1.6. Регистр данных

Этот регистр предназначен для обмена данными между флоппи-дисководом и хостом (компьютером). Использует порт под номером 3F5h. Может применяться как для чтения, так и для записи. Размер регистра данных равен 16 байтам.

11.2.1.7. Регистр цифрового ввода

Регистр использует порт под номером 3F7h. Может применяться только для чтения. Задействован только старший бит (7), а остальные не используются. Если происходит смена диска, бит будет установлен в 1.

11.2.1.8. Регистр управления конфигурацией

Данный регистр позволяет установить скорость передачи данных (см. табл. 11.28). При этом используются только биты 0 и 1. Остальные должны быть равны 0. Использует порт под номером 3F7h. Может применяться только для записи.

11.2.2. Команды управления для флоппи-дисковода

Для управления флоппи-дисководом используется специальный набор команд. Весь процесс выполнения команды можно разделить на три основные фазы: посылка команды, выполнение и получение результата. В первой фазе через регистр данных передается сама команда (строгая последовательность определенных байтов). Далее наступает вторая фаза, когда контроллер выполняет команду. После завершения обработки команды наступает последняя фаза, в которой считывается результат выполнения команды. Список команд показан в табл. 11.30.

Таблица 11.30. *Список команд для флоппи-дисковода*

Имя команды	Описание
READ DATA	Чтение данных
READ DELETED DATA	Чтение удаленных данных
WRITE DATA	Запись данных
WRITE DELETED DATA	Запись удаленных данных

Таблица 11.30 *(окончание)*

Имя команды	Описание
READ TRACK	Чтение дорожки
VERIFY	Проверка данных
FORMAT TRACK	Форматирование дорожки
RECALIBRATE	Рекалибровка устройства
SENSE INTERRUPT STATUS	Получить состояние прерывания
SENSE DRIVE STATUS	Получить состояние флоппи-дисковода
SEEK	Поиск
READ ID	Получить идентификатор

Каждая из представленных команд имеет собственные значения параметров. Для упрощения работы используются специальные сокращения, полный список которых представлен в табл. 11.31.

***Таблица 11.31.** Список принятых сокращений для описания параметров команд*

Сокращение	Описание
AUTO PD	Автоматическое управление энергопотреблением (0 — отключено, 1 — включено)
C	Номер цилиндра (от 0 до 255)
D0, D1	Выбор дисковода от 0 до 3
D	Шаблон данных для форматирования каждого сектора
DIR	Управление движением головки (0 — от шпинделя, 1 — к шпинделю)
DS0, DS1	Выбор дисковода (`00b` — первый, `01b` — второй)
DTL	Размер сектора (если N больше 0, сюда следует записать `FFh`, если N равно 0, сюда записывается количество байтов для чтения или записи)
DRATE [0:1]	Скорость передачи данных в регистре DSR
DRT0, DRT1	Выбор таблицы скорости передачи данных (регистры DSR и CCR)
DT0, DT1	Выбор типа плотности для диска
EC	При установке бита в 1 разрешается счетчик для команды проверки данных VERIFY
EFIFO	Управление работой FIFO (0 — разрешить, 1 — запретить)

Таблица 11.31 (продолжение)

Сокращение	Описание
EIS	Управление поиском (0 — не использовать поиск, 1 — выполнять поиск в фазе выполнения команды перед операцией чтения или записи)
EOT	Конец дорожки (финальный номер сектора для текущей дорожки)
EREG EN	Включение дополнительного регистра состояния (0 — использовать стандартный регистр, 1 — использовать TDR- и SRB-регистры)
GAP	Размер промежутка 2 для перпендикулярного режима
GPL	Размер промежутка 3 для расстояния между секторами
FD0, FD1	Выбор флоппи-дисковода (`00b` — первый, `01b` — второй)
HDS	Номер головки (0 или 1)
HLT	Время готовности к работе головки
HUT	Время удержания головки в рабочем состоянии
Lock	Блокирует параметры EFIFO, PRETRK и FIFOTHR
MFM	Установка бита в 1 включает режим двойной плотности для записи данных
MT	Установка мультидорожечного режима работы (1 — включить)
N	Код размера сектора в байтах (`00h` — 128 байт, `01h` — 256 байт, `02h` — 512 байт, `03h` — 1024 байта, `04h` — 2048 байт, `05h` — 4096 байт, `06h` — 8192 байта, `07h` — 16 384 байта)
NCN	Номер очередного цилиндра для операции поиска
ND	Управление режимом DMA (0 — разрешен, 1 — запрещен)
NRP	Установка бита в 1 позволит отключить фазу результата выполнения команды
PCN	Номер текущего цилиндра, получаемый с помощью команды `SENSE INTERRUPT STATUS`
PC2, PC2, PC0	Значение предкомпенсации в регистре DSR
PDOSC	Если бит равен 1, тактовый генератор отключен
PTS	Выбор предкомпенсации (0 — регистр DSR настроен на определенное время задержки, 1 — задержка не используется)
POLL	Управление режимом внутреннего опроса (0 — включен, 1 — выключен)
PRETRK	Номер стартовой дорожки для предкомпенсации (`00h`—`FFh`)
R	Номер сектора
SC	Количество секторов

Таблица 11.31 (окончание)

Сокращение	Описание
SK	Флаг пропуска (1 — сектора, помеченные как удаленные, будут пропущены; 0 — сектора будут прочитаны или записаны)
SRT	Временной интервал движения головки в миллисекундах (от 0,5 до 8 с шагом 0,5)
STO, ST1, ST2	Внутренние регистры статуса (определяют результат выполнения команды)

А теперь разберем сами команды для работы с флоппи-дисководом и контроллером.

11.2.2.1. Команда *READ DATA*

Эта команда предназначена для чтения данных. Размер команды равен 9 байтам (2 байта описания команды и 7 байтов с параметрами). В третьей фазе команды возвращаются 7 байтов. Формат команды для первой и третьей фаз показан в табл. 11.32.

Таблица 11.32. Команда `READ DATA`

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	MT	MFM	SK	0	0	1	1	0
	0	0	0	0	0	HDS	DS1	DS0
	C							
	H							
	R							
	N							
	EOT							
	GPL							
	DTL							
Третья фаза	ST0							
	ST1							
	ST2							
	C							
	H							
	R							
	N							

Размер блока данных в байтах, которые будут считаны, определяется параметром N. При мультидорожечной операции (бит MT установлен) данные будут считываться одновременно с двух дорожек (табл. 11.33).

Таблица 11.33. *Размеры передаваемых данных*

Количество секторов на дорожке	Размер одного сектора, байт	MT	N	Размер данных, байт
26	256	0	1	6656
52	256	1	1	13312
15	512	0	2	7680
30	512	1	2	15360

После выполнения команды будут возвращены, в том числе, и 3 байта статуса: ST0, ST1, ST2. По ним можно судить о результате выполнения операции. Форматы этих байтов показаны соответственно в табл. 11.34, 11.35 и 11.36.

Таблица 11.34. *Формат байта статуса ST0*

Биты	7	6	5	4	3	2	1	0
Описание	Код завершения (IC)		Поиск (SE)	Ошибка (EC)	Не готов (NR)	Номер головки (HD)	Номер диска (US)	

Приведем краткое описание таблицы.

- Биты 0—1 содержат номер дисковода.
- Бит 2 определяет номер головки дисковода.
- Бит 3 будет установлен в 1, если дисковод не готов к работе.
- Бит 4 указывает на ошибку в работе дисковода, если установлен в 1.
- Бит 5 установлен в 1 после выполнения команды поиска.
- Биты 6—7 определяют код завершения команды:
  - `00b` — успешное завершение;
  - `01b` — команда завершилась ошибкой;
  - `10b` — неправильные параметры или код команды;
  - `11b` — команда не выполнена из-за отсутствия диска в дисководе.

Таблица 11.35. *Формат байта статуса ST1*

Биты	7	6	5	4	3	2	1	0
Описание	Сектор (EN)	0	Ошибка (DE)	Буфер (OR)	0	Нет данных (ND)	Защита (NW)	Маркер (MA)

Приведем описание таблицы.

- Бит 0 установлен в 1, если отсутствует адресная метка (маркер), задаваемая при форматировании диска.
- Бит 1 устанавливается в 1, если выполняется попытка записи на защищенный диск.
- Бит 2 установлен в 1 при отсутствии данных в заданном секторе.
- Бит 4 установлен в 1, если переполнен буфер данных.
- Бит 5 установлен в 1, если возникла ошибка в передаваемых данных.
- Бит 7 установлен в 1, если выполнена попытка обращения к сектору, номер которого больше максимально возможного.

Таблица 11.36. *Формат байта статуса ST2*

Биты	7	6	5	4	3	2	1	0
Описание	0	Метка (CM)	CRC (DD)	Ц (WC)	0	0	СЦ (BC)	Метка (MD)

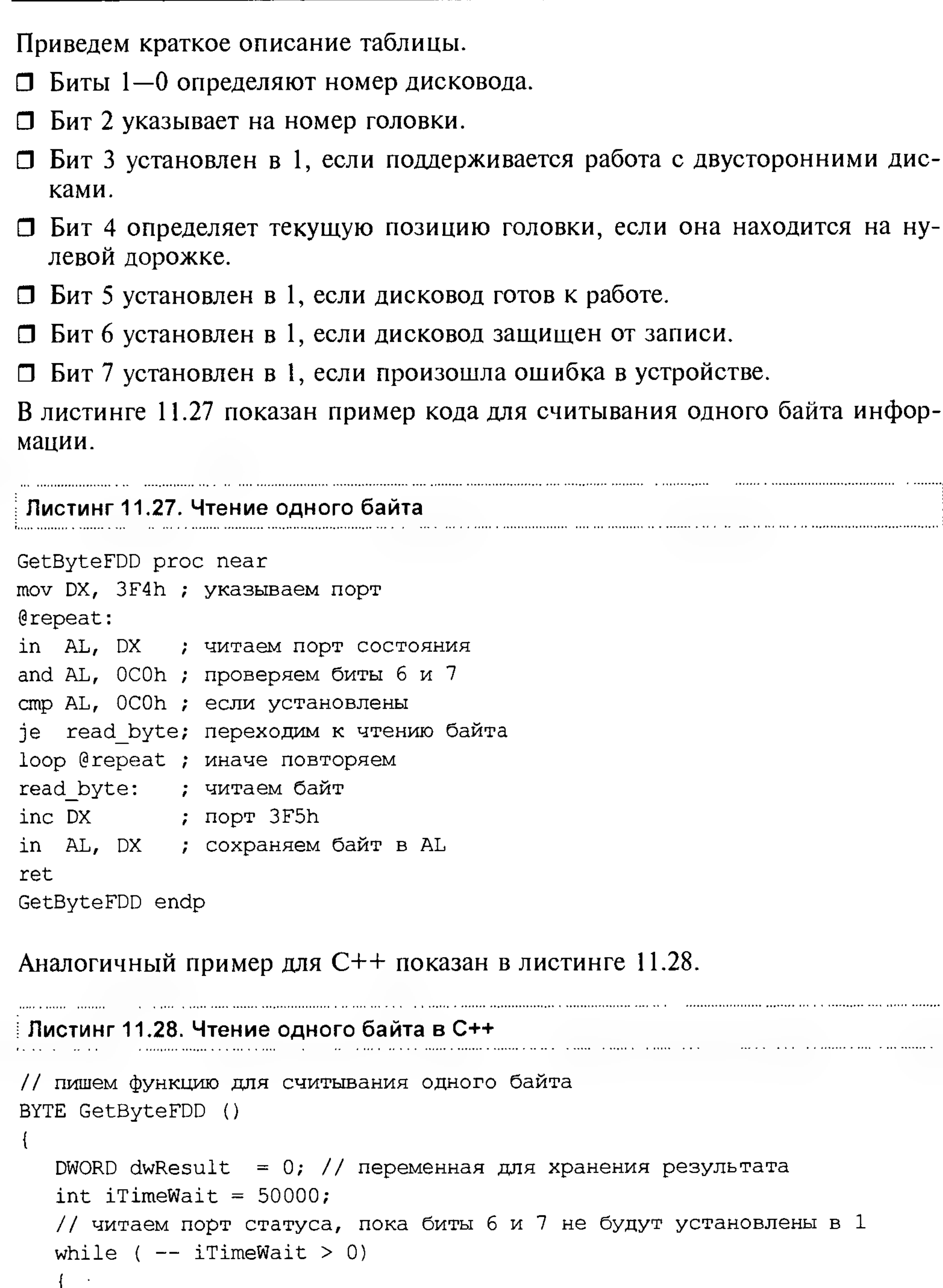
Приведем краткий комментарий к таблице.

- Бит 0 установлен в 1, если отсутствует адресная метка.
- Бит 1 установлен в 1, если обнаружен дефектный цилиндр.
- Бит 4 установлен в 1, если возникла ошибка с определением номера цилиндра.
- Бит 5 установлен в 1, если есть ошибка проверки по контрольной сумме CRC.
- Бит 6 установлен в 1, если попалась метка для удаленных данных.

Дополнительно существует еще один байт статуса (ST3), формат которого представлен в табл. 11.37.

Таблица 11.37. *Формат байта статуса ST3*

Биты	7	6	5	4	3	2	1	0
Описание	Ошибка (FT)	Защита (WP)	Готов (RDY)	ТП (TO)	Число сторон (TS)	Номер головки (HD)	Номер диска (US)	



Приведем краткое описание таблицы.

- Биты 1—0 определяют номер дисковода.
- Бит 2 указывает на номер головки.
- Бит 3 установлен в 1, если поддерживается работа с двусторонними дисками.
- Бит 4 определяет текущую позицию головки, если она находится на нулевой дорожке.
- Бит 5 установлен в 1, если дисковод готов к работе.
- Бит 6 установлен в 1, если дисковод защищен от записи.
- Бит 7 установлен в 1, если произошла ошибка в устройстве.

В листинге 11.27 показан пример кода для считывания одного байта информации.

Листинг 11.27. Чтение одного байта

```
GetByteFDD proc near
mov DX, 3F4h ; указываем порт
@repeat:
in  AL, DX   ; читаем порт состояния
and AL, 0C0h ; проверяем биты 6 и 7
cmp AL, 0C0h ; если установлены
je  read_byte; переходим к чтению байта
loop @repeat ; иначе повторяем
read_byte:   ; читаем байт
inc DX       ; порт 3F5h
in  AL, DX   ; сохраняем байт в AL
ret
GetByteFDD endp
```

Аналогичный пример для C++ показан в листинге 11.28.

Листинг 11.28. Чтение одного байта в C++

```
// пишем функцию для считывания одного байта
BYTE GetByteFDD ()
{
   DWORD dwResult  = 0; // переменная для хранения результата
   int iTimeWait = 50000;
   // читаем порт статуса, пока биты 6 и 7 не будут установлены в 1
   while ( -- iTimeWait > 0)
   {
      // читаем состояние порта
      inPort ( 0x3F4, &dwResult, 1);
```

```
        if ( ( dwResult & 0xC0) == 0xC0) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return ERROR_TIME;
    }
    // читаем байт
    inPort ( 0x3F5, &dwResult, 1);
    return (BYTE) dwResult;
}
```

Для чтения сектора флоппи-диска необходимо выполнить следующую последовательность операций:

1. Инициализировать устройство и включить двигатель.
2. Установить скорость передачи данных или использовать заданную по умолчанию.
3. Подождать не менее 0,5 сек, пока раскрутится двигатель.
4. Провести рекалибровку.
5. Инициализировать контроллер DMA.
6. Передать команду чтения.
7. Включить счетчик времени ожидания.
8. Проверить прерывание от контроллера гибких дисков. Если прерывания нет, увеличить счетчик времени. По истечении времени ожидания сгенерировать ошибку чтения данных и завершить программу.
9. Если прерывание получено, прочитать блок данных (третья фаза). Если операция выполнена, завершить программу.
10. Если операция не выполнена, сделать еще 3 попытки. Каждая попытка должна начинаться с 5-го пункта.
11. Если в течение трех повторений не удалось прочитать данные, следует завершить программу с ошибкой чтения.

Чтобы прочитать сектор диска, можно выполнить код из листинга 11.29.

Листинг 11.29. Чтение сектора диска

```
; выделяем буфер для считываемых данных
data_buffer DB 512 DUP (?)
; и буфер для байта статуса
status_bytes DB 7 DUP (?)
Read_Sector proc near
; запускаем двигатель
mov DX, 3F2h ; указываем порт
mov AL, 0    ; выполняем сброс
```

```
out DX, AL   ; записываем значение в порт
mov AL, 1Ch  ; инициализируем первый дисковод и запускаем двигатель
out DL, AL   ; записываем значение в порт
mov CX, 4000 ; устанавливаем небольшое ожидание для разгона двигателя
@delay:
loop @delay  ; двигатель раскрутится после завершения цикла
; выполняем рекалибровку
call Recalibrate
; устанавливаем начальную позицию
call SetPos
; ждем, пока головка станет в указанную позицию
mov CX, 1800 ; не более 30 мс
@repeat:
loop @repeat
; инициализируем DMA
call InitDMA
; читаем сектор
mov AH, 66h  ; команда READ RATA
call SetByteFDD; записываем первый байт команды
mov AH, 0    ; головка 0 и первый дисковод
call SetByteFDD; записываем второй байт команды
mov AH, 0    ; цилиндр 0
call SetByteFDD; записываем третий байт команды
mov AH, 0    ; головка 0
call SetByteFDD; записываем четвертый байт команды
mov AH, 1    ; сектор 1
call SetByteFDD; записываем пятый байт команды
mov AH, 2    ; размер сектора для 512 байт
call SetByteFDD; записываем шестой байт команды
mov AH, 13h  ; номер последнего сектора для диска 1,44
call SetByteFDD; записываем седьмой байт команды
mov AH, 1Bh  ; длина между секторами
call SetByteFDD; записываем восьмой байт команды
mov AH, 0FFh ; размер сектора не используется
call SetByteFDD; записываем восьмой байт команды
call Wait_INT; ждем завершения команды
; получаем результирующие байты
mov CX, 7    ; должно быть 7 байтов
lea BX, status_bytes; получаем адрес буфера
@get_bytes:
call GetByteFDD      ; читаем байт
mov[BX], AL  ; записываем в очередную позицию status_bytes
inc BX       ; переходим на следующий байт
loop @get_bytes; повторяем 7 раз
```

```
; глушим двигатель
mov DX, 3F2h ; указываем порт
mov AL, 0Ch  ; останавливаем двигатель
out DL, AL   ; записываем значение в порт
ret
Read_Sector endp
; рекалибровка дисковода
Recalibrate proc near
mov AL, 7    ; команда RECALIBRATE
call SetByteFDD; записываем первый байт команды
mov AL, 0    ; первый дисковод (A)
call SetByteFDD; записываем второй байт команды
call Wait_INT; ждем завершения команды
ret
Recalibrate endp
; процедура для ожидания прерывания дисковода
Wait_INT proc near; прерывание INT6
mov AX, 40h  ; данные BIOS
mov ES, AX   ; заносим в ES
mov BX, 3Eh  ; определяем байт статуса
@repeat:     ; проверяем бит 7
mov DL, ES:[BX]; копируем байт статуса
test DL, 80h ; если бит 7 не равен 1
jz  @repeat  ; повторяем опрос
and DL, 7Fh  ; обнуляем бит 7
mov ES:[BX], DL; обновляем байт статуса
ret
Wait_INT endp
; процедура для установки позиции
SetPos proc near
mov AH, 0Fh  ; команды SEEK
call SetByteFDD; записываем первый байт команды
mov AL, 0    ; первый дисковод (A)
call SetByteFDD; головка с номером 0, первый дисковод (A)
mov AH, 0    ; номер цилиндра для операции поиска
call SetByteFDD; записываем третий байт команды
call Wait_INT; ждем завершения команды
ret
SetPos endp
; инициализация DMA
InitDMA proc near
cli          ; запрещаем прерывания
mov AL, 46h  ; команда READ DATA
out 12, AL
out 11, AL
```

```
mov AX, offset data_buffer; адрес буфера данных
mov BX, DS
mov CL, 4
rol BX, CL    ; сдвигаем влево на 4 бита
mov DL, BL
and BL, 0F0h ; обнуляем младшие 4 бита
and DL, 0Fh  ; обнуляем старшие 4 бита
add AX, BX   ; суммируем
jnc @dalee   ; если CF = 0
inc DL       ; увеличиваем DL
@dalee:
out 4, AL    ; пишем младший байт смещения адреса
mov AL, AH   ; старший байт смещения адреса
out 4, AL    ; пишем старший байт
mov AL, DL   ; страница буфера DMA
out 81h, AL  ; записываем номер страницы
mov AX, 511  ; устанавливаем размер данных
out 5, AL    ; пишем младший байт
mov AL, AH   ; старший байт
out 5, AL    ; пишем старший байт
mov AL, 2    ; выделяем канал 2
out 10, AL   ; пишем выбранный канал DMA
sti          ; разрешаем прерывания
ret
InitDMA endp
```

11.2.2.2. Команда *READ DELETED DATA*

Данная команда предназначена для чтения данных, в том числе и удаленных. Размер команды равен 9 байтам (2 байта описания команды и 7 байтов с параметрами). В третьей фазе команды возвращаются 7 байтов. Команда аналогична команде `READ DATA`, за исключением первого байта (табл. 11.38).

Таблица 11.38. *Формат команды* `READ DELETED DATA`

Биты	7	6	5	4	3	2	1	0
Фаза 1	MT	MFM	SK	0	1	1	0	0

Последовательность работы с данной командой не отличается от обычной команды чтения.

11.2.2.3. Команда *WRITE DATA*

Команда позволяет записать данные на диск. Размер команды равен 9 байтам (2 байта описания команды и 7 байтов с параметрами). В третьей

фазе команды возвращаются 7 байтов. Формат команды совпадает с READ DATA, за исключением первого байта (табл. 11.39).

Таблица 11.39. *Формат команды* WRITE DATA

Биты	7	6	5	4	3	2	1	0
Фаза 1	MT	MFM	0	0	0	1	0	1

Для записи байта в порт можно использовать код, приведенный в листинге 11.30.

Листинг 11.30. Запись одного байта

```
SetByteFDD proc near
mov DX, 3F4h ; указываем порт
@repeat:
in  AL, DX   ; читаем порт состояния
test AL, 80h ; проверяем бит 7
jz  @repeat  ; если бит 7 не равен 1, повторяем
inc DX       ; порт 3F5h
mov AL, AH   ; передаем байт
out DX, AL   ; пишем байт в порт
ret
SetByteFDD endp
```

Аналогичный пример для C++ показан в листинге 11.31.

Листинг 11.31. Запись одного байта в C++

```
// пишем функцию для записи одного байта
bool SetByteFDD ( BYTE Data)
{
   DWORD dwResult  = 0; // переменная для хранения результата
   int iTimeWait = 50000;
   // читаем порт статуса, пока бит 7 не будет установлен в 1
   while ( -- iTimeWait > 0)
   {
       // читаем состояние порта
       inPort ( 0x3F4, &dwResult, 1);
       if ( ( dwResult & 0x80) == 0x01) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return false;
   }
```

```
    // пишем байт
    outPort ( 0x3F5, Data, 1);
    return true;
}
```

11.2.2.4. Команда *WRITE DELETED DATA*

Эта команда позволяет записать данные на диск. Каждый записываемый сектор помечается как удаленный. Размер команды равен 9 байтам (2 байта описания команды и 7 байтов с параметрами). В третьей фазе команды возвращаются 7 байтов. Формат команды совпадает с READ DATA, за исключением первого байта (табл. 11.40).

***Таблица 11.40.** Формат команды* WRITE DELETED DATA

Биты	7	6	5	4	3	2	1	0
Фаза 1	MT	MFM	0	0	1	0	0	1

11.2.2.5. Команда *READ TRACK*

Команда позволяет прочитать данные для указанной дорожки. Размер команды равен 9 байтам (2 байта описания команды и 7 байтов с параметрами). В третьей фазе команды возвращаются 7 байтов. Формат команды совпадает с READ DATA, за исключением первого байта (табл. 11.41).

***Таблица 11.41.** Формат команды* READ TRACK

Биты	7	6	5	4	3	2	1	0
Фаза 1	0	MFM	0	0	0	0	1	0

11.2.2.6. Команда *VERIFY*

Эта команда позволяет проверить данные на диске. Размер команды равен 9 байтам (2 байта описания команды и 7 байтов с параметрами). В третьей фазе команды возвращаются 7 байтов. Формат команды показан в табл. 11.42.

11.2.2.7. Команда *FORMAT TRACK*

Данная команда позволяет отформатировать заданный носитель. Размер команды равен 6 байтам (2 байта описания команды и 4 байта с параметрами). Во второй фазе возвращается информация о форматировании текущего сектора. В третьей фазе команды возвращаются 7 байтов. Формат команды показан в табл. 11.43.

Таблица 11.42. Команда `VERIFY`

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	MT	MFM	SK	1	0	1	1	0
	EC	0	0	0	0	HDS	DS1	DS0
	C							
	H							
	R							
	N							
	EOT							
	GPL							
	DTL / SC							
Третья фаза	ST0							
	ST1							
	ST2							
	C							
	H							
	R							
	N							

Таблица 11.43. Команда `FORMAT TRACK`

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	0	MFM	0	0	1	1	0	1
	0	0	0	0	0	HDS	DS1	DS0
	N							
	SC							
	GPL							
	D							

Таблица 11.43 (окончание)

Фаза	Биты							
	7	6	5	4	3	2	1	0
Вторая фаза	C							
	H							
	R							
	N							
Третья фаза	ST0							
	ST1							
	ST2							
	Не определен							
	Не определен							
	Не определен							
	Не определен							

В табл. 11.44 приводятся стандартные значения для некоторых полей команды.

***Таблица 11.44.** Стандартные значения для форматирования дисков*

Тип	Размер	Размер сектора, байт	N	SC	GPL (чтение и запись)	GPL (формат)
5,25"	360 Кб	512	02h	09h	2Ah	50h
	1,2 Мб	512	02h	0Fh	2Ah	50h
3,5"	720 Кб	512	02h	09h	1Bh	54h
	1,44 Мб	512	02h	18h	1Bh	54h
	2,88 Мб	512	02h	24h	38h	53h

11.2.2.8. Команда *RECALIBRATE*

Эта команда позволяет установить головки дисковода в нулевую позицию. Размер команды равен 2 байта. Третья фаза отсутствует. Формат команды показан в табл. 11.45. После выполнения данной команды следует вызвать команду `SENSE INTERRUPT STATUS` для получения результатов.

Таблица 11.45. Команда `RECALIBRATE`

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	0	0	0	0	0	1	1	1
	0	0	0	0	0	0	DS1	DS0

В листинге 11.32 показан пример кода, выполняющий рекалибровку для первого дисковода (A:).

Листинг 11.32. Выполнение рекалибровки флоппи-дисковода

```
; рекалибровка дисковода
mov AL, 7    ; команда RECALIBRATE
call SetByteFDD; записываем первый байт команды
mov AL, 0    ; первый дисковод (A:)
call SetByteFDD; записываем второй байт команды
call GetWaitINT; ждем прерывания
```

Этот же пример для C++ представлен в листинге 11.33.

Листинг 11.33. Выполнение рекалибровки флоппи-дисковода в C++

```
// функция рекалибровки дисковода
void RecalibrateFDD ( unsigned int uDisk)
{
   SetByteFDD ( 0x07); // команда RECALIBRATE
   SetByteFDD ( (BYTE) uDisk); // первый дисковод (A:)
   GetWaitINT (); // ждем прерывания
}
```

11.2.2.9. Команда *SENSE INTERRUPT STATUS*

Данная команда позволяет получить информацию о состоянии прерывания для флоппи-дисковода. Размер команды равен 1 байту. В третьей фазе возвращаются 2 байта. Формат команды показан в табл. 11.46.

Контроллер дисковода вырабатывает сигнал прерывания в следующих случаях:

- после выполнения команды `READ DATA` (третья фаза);
- после выполнения команды `READ DELETED DATA` (третья фаза);
- после выполнения команды `READ TRACK` (третья фаза);

- после выполнения команды READ ID (третья фаза);
- после выполнения команды WRITE DATA (третья фаза);
- после выполнения команды WRITE DELETED DATA (третья фаза);
- после выполнения команды FORMAT TRACK (третья фаза);
- после выполнения команды SEEK;
- после выполнения команды RECALIBRATE;
- при передаче данных не в DMA режиме.

Таблица 11.46. Формат команды SENSE INTERRUPT STATUS

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	0	0	0	0	1	0	0	0
Третья фаза	ST0 PCN							

В результате выполнения команды SENSE INTERRUPT STATUS будет возвращен байт состояния ST0 (см. табл. 11.34). Причина прерывания и результат будут закодированы в битах IC (6 и 7) и SE (5). Если сигнал прерывания отсутствует, команда запишет в ST0 значение 80h. Рассмотрим пример для получения информации о прерывании, показанный в листинге 11.34.

Листинг 11.34. Получение информации о прерывании

```
GetWaitINT proc near
mov AL, 8     ; команда SENSE INTERRUPT STATUS
call SetByteFDD; записываем первый байт команды
@repeat:
call GetByteFDD; записываем первый байт команды
and AL, 80h   ; проверяем биты 6 и 7
cmp AL, 80h   ; если прерывания нет
je  @repeat   ; повторяем опрос
ret
GetWaitINT endp
```

Аналогичный код для C++ приведен в листинге 11.35.

Листинг 11.35. Получение информации о прерывании в C++

```
// пишем функцию для получения статуса прерывания
void GetWaitINT ()
{
   DWORD dwResult  = 0; // переменная для хранения результата
   int iTimeWait = 50000;
   // команда SENSE INTERRUPT STATUS
   SetByteFDD ( 0x08);
   // читаем порт 3F5h
   while ( -- iTimeWait > 0)
   {
       inPort ( 0x3F5, &dwResult, 1);
       // если прерывания есть, выходим из функции
       if ( dwResult & 0x80) == 0x80) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return ERROR_TIME;
   }
}
```

11.2.2.10. Команда *SENSE DRIVE STATUS*

Эта команда позволяет получить информацию о состоянии дисковода. Размер команды равен 1 байту. В третьей фазе возвращается 1 байт. Формат команды показан в табл. 11.47.

Таблица 11.47. *Формат команды* SENSE DRIVE STATUS

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	0	0	0	0	0	1	0	0
Третья фаза	ST3							

11.2.2.11. Команда *SEEK*

Данная команда позволяет переместить головки дисковода в заданную позицию (цилиндр). Размер команды равен 3 байта. После начала выполнения команды сравнивается заданное значение цилиндра NCN с текущим PCN. Если значения не совпадают, контроллер производит пошаговый поиск с проверкой результата после каждого шага. Формат команды показан в табл. 11.48. После выполнения данной команды следует вызвать SENSE INTERRUPT STATUS для получения результатов.

Таблица 11.48. Формат команды SEEK

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	0	0	0	0	1	1	1	1
Вторая фаза	—							

11.2.2.12. Команда *READ ID*

Команда позволяет получить текущую позицию головок дисковода. Размер команды равен 2 байта. Формат команды показан в табл. 11.49.

Таблица 11.49. Команда READ ID

Фаза	Биты							
	7	6	5	4	3	2	1	0
Первая фаза	0	MFM	0	0	1	0	1	0
	0	0	0	0	0	HDS	DS1	DS0
Третья фаза	ST0 ST1 ST2 C H R N							

На этом завершим описание команд для флоппи-дисководов и перейдем к программированию устройств с интерфейсом ATA (AT Attachment Interface) и ATAPI (AT Attachment with Packet Interface Extension).

11.2.3. Устройства ATA/ATAPI

Интерфейс ATAPI отличается от ATA способом передачи данных: запись и считывание информации происходят в пакетном режиме, похожим на работу устройств SCSI. Интерфейс ATA позволяет управлять четырьмя дисководами, подключенными к контроллеру через два канала. На каждом канале

одно из устройств является ведущим (Device 0), а другое — ведомым (Device 1). К устройствам ATA относятся в основном все современные жесткие диски, а к ATAPI — такие устройства, как CD-ROM, CD-RW, DVD-ROM, DVD-RW и т. п.

Для работы с устройствами ATA и ATAPI существуют жестко закрепленные порты ввода-вывода и номера прерываний (табл. 11.50).

Таблица 11.50. *Доступ к устройствам ATA и ATAPI*

Канал	Номер прерывания	Управляющие регистры	Дополнительные регистры
1	14	1F0—1F7 (8 байт)	3F6h (1 байт)
2	15	170–177 (8 байт)	376h (1 байт)

Каждому каналу выделены собственные номера регистров. Назначение этих регистров описано в табл. 11.51.

Таблица 11.51. *Назначение регистров*

Канал 1	Канал 2	Использование регистра	
		Чтение	Запись
1F0h	170h	Регистр данных (DR)	—
1F1h	171h	Регистр ошибки (ER)	Регистр особенностей (FT)
1F2h	172h	Регистр счетчика секторов (SC)	Регистр счетчика секторов (SC)
1F3h	173h	Регистр номера сектора (SN)	Регистр номера сектора (SN)
1F4h	174h	Регистр младшего байта номера цилиндра (CL)	Регистр младшего байта номера цилиндра (CL)
1F5h	175h	Регистр старшего байта номера цилиндра (CH)	Регистр старшего байта номера цилиндра (CH)
1F6h	176h	Регистр выбора устройства и номера головки (DH)	—
1F7h	177h	Регистр состояния (SR)	Регистр команд (CR)
3F6h	376h	Дополнительный регистр состояния (AC)	Регистр управления (DC)

Как видно из таблицы, некоторые регистры имеют двойное назначение, в зависимости от выполняемой операции (запись или чтение). К сожалению, адресация CHS последнее время не используется. Ее полностью заменил

режим логической адресации LBA. Для указания адреса сектора применяется 28-разрядный адрес LBA. Исходя из этого, изменилось и назначение регистров. В табл. 11.52 и 11.53 приводится новое назначение регистров для устройств ATA и ATAPI.

Таблица 11.52. *Назначение регистров для ATA*

Канал 1	Канал 2	Использование регистра	
		Чтение	**Запись**
1F0h	170h	Регистр данных (DR)	Регистр данных (DR)
1F1h	171h	Регистр ошибки (ER)	Регистр особенностей (FT)
1F2h	172h	Регистр счетчика секторов (SC)	Регистр счетчика секторов (SC)
1F3h	173h	Первый байт адреса (0—7)	Первый байт адреса (0—7)
1F4h	174h	Второй байт адреса (8—15)	Второй байт адреса (8—15)
1F5h	175h	Третий байт адреса (16—23)	Третий байт адреса (16—23)
1F6h	176h	Регистр выбора устройства и 4 бита LBA адреса (24—27)	Регистр выбора устройства и 4 бита LBA адреса (24—27)
1F7h	177h	Регистр состояния (SR)	Регистр команд (CR)
3F6h	376h	Дополнительный регистр состояния (AC)	Регистр управления устройством (DC)

Таблица 11.53. *Назначение регистров для ATAPI*

Канал 1	Канал 2	Использование регистра	
		Чтение	**Запись**
1F0h	170h	Регистр данных (DR)	Регистр данных (DR)
1F1h	171h	Регистр ошибки (ER)	Регистр особенностей (FT)
1F2h	172h	Регистр прерывания	—
1F3h	173h	—	—
1F4h	174h	Младший байт пакета данных	Младший байт пакета данных
1F5h	175h	Старший байт пакета данных	Старший байт пакета данных
1F6h	176h	Выбор устройства	Выбор устройства
1F7h	177h	Регистр состояния (SR)	Регистр команд (CR)

Рассмотрим назначение каждого регистра подробнее.

11.2.3.1. Регистр данных

Данный регистр используется для передачи данных в операциях чтения и записи. Это единственный из всех регистров, имеющий размер 16 бит, однако дополнительные 8 бит он "уводит" у регистра ошибок, перекрывая последний. При этом младший байт расположен в регистре `1x0h`, а старший — в `1x1h`.

11.2.3.2. Регистр ошибки

Регистр доступен для считывания и позволяет получить результат выполненной операции. Формат регистра ошибки может меняться в зависимости от выполненной команды. Не изменяется только значение бита 2 (ABRT). Если он установлен в 1, значит, команда была прервана из-за ошибки. Проверку этого регистра следует проводить только тогда, когда бит 0 (ERR) в регистре состояния установлен в 1. Для устройств ATAPI формат регистра показан в табл. 11.54.

Таблица 11.54. *Формат регистра ошибки для ATAPI*

Биты	7	6	5	4	3	2	1	0
Описание	Sense Key				—	ABRT	EOM	ILI

Приведем краткое описание таблицы.

- Бит 0, установленный в 1, означает, что указан неправильный размер пакета команды или блока с данными.
- Бит 1, установленный в 1, определяет, что найден конец носителя.
- Бит 2, установленный в 1, означает, что команда была прервана из-за ошибки.
- Биты 4—7 определяют значение ключа смысла, описывающего ошибку.

11.2.3.3. Регистр особенностей

Этот регистр доступен только для записи. Используется для выполнения различных установок и зависит от используемой команды. Для устройств ATAPI формат регистра показан в табл. 11.55.

Таблица 11.55. *Формат регистра особенностей для ATAPI*

Биты	7	6	5	4	3	2	1	0
Описание	Резерв						OVL	DMA

Приведем краткий комментарий к таблице.

- Бит 0 управляет выбором режима передачи: 1 — DMA, 0 — PIO.
- Бит 1 управляет выбором режима перекрытия команд: 1 — включен, 0 — выключен.

11.2.3.4. Регистр счетчика секторов

Данный регистр определяет общее количество секторов, которое будет записано или считано. Регистр доступен для записи и чтения. При передаче очередного сектора значение в этом регистре уменьшается на 1, что позволяет контролировать число реально обработанных секторов. Для этого достаточно прочитать значение из регистра.

11.2.3.5. Регистр прерывания

Регистр позволяет определить различную информацию о состоянии прерывания. Доступен только для чтения. Бит 0 (C/D) определяет тип данных (1 — управляющая команда, 0 — пользовательские данные). Бит 1 (I/O) указывает направление передачи данных (1 — от устройства к компьютеру, 0 — от компьютера к устройству). Бит 2 (REL) помогает определить, свободна ли шина (1 — шина свободна).

11.2.3.6. Регистр номера сектора

Этот регистр позволяет установить начальный сектор для операций записи или считывания. Доступен для записи и чтения. При использовании логической адресации содержит младший байт адреса (0—7).

11.2.3.7. Регистр младшего байта номера цилиндра

Регистр позволяет установить начальный номер цилиндра (младший байт). Доступен для записи и чтения. При использовании логической адресации содержит второй байт адреса (8—15).

11.2.3.8. Регистр старшего байта номера цилиндра

Данный регистр позволяет установить начальный номер цилиндра (старший байт). Доступен для записи и чтения. При использовании логической адресации содержит второй байт адреса (16—23).

11.2.3.9. Регистр выбора устройства и номера головки

Регистр позволяет выбрать номер устройства (0 или 1), а также дополнительные параметры в зависимости от режима адресации. Формат регистра показан в табл. 11.56.

Таблица 11.56. Формат регистра выбора устройства

Биты	7	6	5	4	3	2	1	0
Описание	1	LBA	1	DEV	x	x	x	x

Приведем краткое описание таблицы.

- Биты 0—3 должны определять номер головки при использовании CHS-адресации. В режиме LBA они должны определять четыре старших разряда адреса (24—27): 0 — 24, 1 — 25, 2 — 26 и 3 — 27.
- Бит 4 позволяет выбрать номер устройства на канале: 1 — ведущее (Master), 0 — ведомое (Slave).
- Бит 5 устарел. Может быть установлен в 1 для совместимости со старыми стандартами.
- Бит 6 позволяет выбрать режим адресации: 0 — CHS, 1 — LBA.
- Бит 7 устарел. Может быть установлен в 1 для совместимости со старыми стандартами.

11.2.3.10. Регистр состояния

Регистр позволяет получить текущее состояние устройства. Доступен только для чтения. Если бит 7 установлен в 1, содержание регистра игнорируется. Формат регистра показан в табл. 11.57.

Таблица 11.57. Формат регистра состояния

Биты	7	6	5	4	3	2	1	0
Описание	BSY	DRDY	DF	x	DRQ	—	—	ERR

Приведем краткое описание таблицы.

- Бит 0, установленный в 1, означает, что произошла ошибка при выполнении команды. Для устройств ATAPI этот бит называется CHK.
- Биты 1—2 устарели и не должны использоваться.
- Бит 3, установленный в 1, определяет, что устройство готово к передаче данных.
- Бит 4 может менять свое назначение в зависимости от выполняемой команды.
- Бит 5, установленный в 1, означает, что произошла неизвестная ошибка, например аппаратный сбой устройства.

- Бит 6 равен 1, когда устройство может выполнить любую поддерживаемую команду.
- Бит 7, установленный в 1, означает, что устройство занято.

11.2.3.11. Регистр команд

Данный регистр позволяет передать устройству определенную команду. Доступен только для записи. Запись команды должна производиться, когда очищены биты BSY и DRQ в регистре состояния. Выполнение команды начинается сразу после записи в этот регистр, поэтому если команда имеет дополнительные параметры, вначале следует записать их в соответствующие регистры.

11.2.3.12. Дополнительный регистр состояния

Регистр содержит ту же информацию, что и регистр состояния. Доступен только для чтения. Если бит 7 установлен в 1, содержание регистра игнорируется. Чтение регистра не сбрасывает сигнал прерывания.

11.2.3.13. Регистр управления

Этот регистр позволяет управлять программным сбросом устройств и сигналом прерывания. Доступен только для записи. Любые изменения в регистре сразу же вступают в силу. Формат регистра показан в табл. 11.58.

Таблица 11.58. Формат регистра состояния

Биты	7	6	5	4	3	2	1	0
Описание	HOB	Резерв	Резерв	Резерв	Резерв	SRST	nIEN	0

Приведем краткое описание таблицы.

- Бит 0 должен быть установлен в 0.
- Бит 1 управляет сигналом прерывания: 1 — запрещен, 0 — разрешен.
- Бит 2 управляет программным сбросом устройств: 1 — выполнить сброс.
- Бит 7 является старшим разрядом для 48-разрядного адреса набора особенностей. Запись любого значения в регистр команд очищает этот бит.

Теперь, когда мы разобрались с имеющимися регистрами, пора познакомиться с управляющими командами. Все команды можно разделить на две группы: обычные и пакетные. К последней группе также можно отнести набор команд SCSI (Multimedia Commands). Здесь мы рассмотрим только команды ATA/ATAPI, а с набором команд SCSI вы можете познакомиться в книге [3].

11.2.4. Команды управления для устройств ATA/ATAPI

Обычные команды могут быть трех видов: обязательные для всех устройств, необязательные и определяемые производителем. Список команд для интерфейса ATA/ATAPI-7 представлен в табл. 11.59.

Таблица 11.59. Список команд ATA/ATAPI-7

Имя команды	Код	Поддержка
CFA ERASE SECTORS	C0h	Необязательная
CFA REQUEST EXTENDED ERROR	03h	Необязательная
CFA TRANSLATE SECTOR	87h	Необязательная
CFA WRITE MULTIPLE WITHOUT ERASE	CDh	Необязательная
CFA WRITE SECTORS WITHOUT ERASE	38h	Необязательная
CHECK MEDIA CARD TYPE	D1h	Необязательная
CHECK POWER MODE	E5h	Обязательная
CONFIGURE STREAM	51h	Необязательная
DEVICE CONFIGURATION FREEZE LOCK	B1h	Необязательная
DEVICE CONFIGURATION IDENTIFY	B1h	Необязательная
DEVICE CONFIGURATION RESTORE	B1h	Необязательная
DEVICE CONFIGURATION SET	B1h	Необязательная
DEVICE RESET	08h	Обязательная для ATAPI
DOWNLOAD MICROCODE	92h	Необязательная
EXECUTE DEVICE DIAGNOSTIC	90h	Обязательная
FLUSH CACHE	E7h	Обязательная для ATA
FLUSH CACHE EXT	EAh	Необязательная
GET MEDIA STATUS	DAh	Необязательная
IDENTIFY DEVICE	ECh	Обязательная для ATA
IDENTIFY PACKET DEVICE	A1h	Обязательная для ATAPI
IDLE	E3h	Обязательная для ATA
IDLE IMMEDIATE	E1h	Обязательная
MEDIA EJECT	EDh	Необязательная
MEDIA LOCK	DEh	Необязательная
MEDIA UNLOCK	DFh	Необязательная

Таблица 11.59 (продолжение)

Имя команды	Код	Поддержка
NOP	00h	Обязательная для ATAPI
PACKET	A0h	Обязательная для ATAPI
READ BUFFER	E4h	Необязательная
READ DMA	C8h	Обязательная для ATA
READ DMA EXT	25h	Необязательная
READ DMA QUEUED	C7h	Необязательная
READ DMA QUEUED EXT	26h	Необязательная
READ LOG EXT	2Fh	Необязательная
READ MULTIPLE	C4h	Обязательная для ATA
READ MULTIPLE EXT	29h	Необязательная
READ NATIVE MAX ADDRESS	F8h	Необязательная
READ NATIVE MAX ADDRESS EXT	27h	Необязательная
READ SECTOR(S)	20h	Обязательная
READ SECTOR(S) EXT	24h	Необязательная
READ STREAM DMA	2Ah	Необязательная
READ STREAM PIO	2Bh	Необязательная
READ VERIFY SECTOR(S)	40h	Обязательная для ATA
READ VERIFY SECTOR(S) EXT	42h	Необязательная
SECURITY DISABLE PASSWORD	F6h	Необязательная
SECURITY ERASE PREPARE	F3h	Необязательная
SECURITY ERASE UNIT	F4h	Необязательная
SECURITY FREEZE LOCK	F5h	Необязательная
SECURITY SET PASSWORD	F1h	Необязательная
SECURITY UNLOCK	F2h	Необязательная
SERVICE	A2h	Необязательная
SET FEATURES	EFh	Обязательная
SET MAX ADDRESS	F9h	Необязательная
SET MAX ADDRESS EXT	37h	Необязательная
SET MULTIPLE MODE	C6h	Обязательная для ATA
SLEEP	E6h	Обязательная
SMART DISABLE OPERATIONS	B0h	Необязательная

Таблица 11.59 (окончание)

Имя команды	Код	Поддержка
SMART ENABLE/DISABLE AUTOSAVE	B0h	Необязательная
SMART ENABLE OPERATIONS	B0h	Необязательная
SMART EXECUTE OFF_LINE IMMEDIATE	B0h	Необязательная
SMART READ DATA	B0h	Необязательная
SMART READ LOG	B0h	Необязательная
SMART RETURN STATUS	B0h	Необязательная
SMART WRITE LOG	B0h	Необязательная
STANDBY	E2h	Обязательная для ATA
STANDBY IMMEDIATE	E0h	Обязательная
WRITE BUFFER	E8h	Необязательная
WRITE DMA	CAh	Обязательная для ATA
WRITE DMA EXT	35h	Необязательная
WRITE DMA FUA EXT	3Dh	Необязательная
WRITE DMA QUEUED	CCh	Необязательная
WRITE DMA QUEUED EXT	36h	Необязательная
WRITE DMA QUEUED FUA EXT	3Eh	Необязательная
WRITE LOG EXT	3Fh	Необязательная
WRITE MULTIPLE	C5h	Обязательная для ATA
WRITE MULTIPLE EXT	39h	Необязательная
WRITE MULTIPLE FUA EXT	CEh	Необязательная
WRITE SECTOR(S)	30h	Обязательная для ATA
WRITE SECTOR(S) EXT	34h	Необязательная
WRITE STREAM DMA	3Ah	Необязательная
WRITE STREAM PIO	3Bh	Необязательная

Мы не будем рассматривать все имеющиеся команды, а разберем только обязательные для всех устройств.

11.2.4.1. Команда *CHECK POWER MODE*

Эта команда предназначена для получения информации о питании устройства. Команда является обязательной для обычных и пакетных устройств. Формат команды показан в табл. 11.60.

Таблица 11.60. Формат команды `CHECK POWER MODE`

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	E5h							

Для инициализации команды следует в регистр `1x6h` записать номер устройства DEV (0 или 1), а затем в регистр `1x7h` записать код команды `E5h`. При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x2h` будет хранить результат команды: `00h` — дежурный режим (Standby), `80h` — режим простоя (Idle), `FFh` — режим простоя (Idle) или активный (Active) режим.
2. Регистр `1x6h` будет хранить номер выбранного устройства.
3. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки регистры будут иметь следующий вид:

1. В регистре `1x1h` бит ABRT будет установлен в 1, если устройство не поддерживает команду или произошло аварийное завершение.
2. Регистр `1x6h` будет хранить номер выбранного устройства.
3. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), DRQ = 0 и ERR = 1.

Например, чтобы проверить состояние питания для жесткого диска на первом канале, можно использовать код из листинга 11.36.

Листинг 11.36. Проверка состояния питания жесткого диска

```
; ждем готовности устройства
IsReadATA proc near
mov DX, 1F7h ; указываем порт
@repeat:
in  AL, DX    ; читаем порт состояния
test AL, 80h ; проверяем бит 7
```

```
jnz @repeat  ; если бит 7 равен 1, повторяем
ret
IsReadATA endp
; наш код
call IsReadATA; ждем, пока порт освободится
dec DX       ; порт 1F6h
mov AL, 0A0h ; первое устройство на первом канале
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F7h
mov AL, 0E5h ; код команды CHECK POWER MODE
out DX, AL   ; записываем значение в порт
call IsReadATA; ждем, пока порт освободится
in  AL, DX   ; читаем регистр статуса
test AL, 01b ; если ошибка
jnz ERROR_HND; передаем управление обработчику
call IsReadATA; ждем, пока порт освободится
mov DX, 1F2h ; выбираем регистр с результатом операции
in  AL, DX   ; читаем значение
cmp AL, 0    ; дежурный режим ?
je  STANDBY_MODE
cmp AL, 80h  ; режим простоя ?
je  IDLE_MODE
cmp AL, FFh  ; неопределенный режим ?
je  ACTIVE_MODE
```

Аналогичный пример для C++ показан в листинге 11.37.

Листинг 11.37. Проверка состояния питания жесткого диска в C++

```
// пишем функцию для ожидания готовности устройства
void IsReadyATA ()
{
   DWORD dwResult  = 0; // переменная для хранения результата
   int iTimeWait = 50000;
   // читаем порт 1F7h
   while ( -- iTimeWait > 0)
   {
       inPort ( 0x1F7, &dwResult, 1);
       // если бит 7 равен 0, выходим
       if ( dwResult & 0x80) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return ERROR_TIME;
   }
}
```

```
// получаем состояние питания
int GetPowerMode ()
{
   DWORD dwResult  = 0; // переменная для хранения результата
   IsReadyATA (); // ждем, пока порт освободится
   outPort ( 0x1F6, 0xA0, 1); // первое устройство на первом канале
   outPort ( 0x1F7, 0xE5, 1); // код команды CHECK POWER MODE
   IsReadyATA (); // ждем, пока порт освободится
   inPort ( 0x1F7, &dwResult, 1); // читаем регистр статуса
   if ( ( dwResult & 0x01) == 0x01) // произошла ошибка
       return -1;
   IsReadyATA (); // ждем, пока порт освободится
   inPort ( 0x1F2, &dwResult, 1); // выбираем регистр
                                  // с результатом операции
   switch ( dwResult)
   {
    case 0: // дежурный режим
      return 1;
    case 128: // режим простоя
      return 2;
    case 256: // активный или простой
      return 3;
   }
}
```

11.2.4.2. Команда *DEVICE RESET*

Команда позволяет выполнить сброс выбранного устройства на шине. Используется только для ATAPI-устройств. Формат команды показан в табл. 11.61.

Таблица 11.61. *Формат команды* DEVICE RESET

Регистры	Биты							
	7	6	5	4	3	2	1	0
1: 1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	08h							

Для инициализации команды следует в регистр 1x6h записать номер устройства DEV (0 или 1), а затем в регистр 1x7h записать код команды 08h. При успешном выполнении регистры будут содержать следующую информацию:

1. В регистр 1x1h будет помещен диагностический код (табл. 11.62).
2. В регистры 1x2h—1x6h будет записана сигнатура (табл. 11.63).
3. В регистр 1x7h будет записана информация в соответствии с протоколом команды.

Таблица 11.62. *Диагностические коды*

Код	Описание
	Первое устройство (Device 0)
01h	Device 0 выполнило команду, Device 1 выполнило команду или отсутствует
00h, 02h—7Fh	Сбой в устройстве Device 0, Device 1 выполнило команду или отсутствует
81h	Device 0 выполнило команду, сбой в устройстве Device 1
80h, 82h—FFh	Сбой в обоих устройствах
	Второе устройство (Device 1)
01h	Device 1 выполнило команду
00h, 02h—7Fh	Сбой в устройстве Device 1

Таблица 11.63. *Значения для сигнатуры*

Регистры	ATA	ATAPI
1x2h	01h	01h
1x3h	01h	01h
1x4h	00h	14h
1x5h	00h	EBh
1x6h	00h	00h (Device 0) или 10h (Device 1)

11.2.4.3. Команда *EXECUTE DEVICE DIAGNOSTIC*

Данная команда позволяет выполнить диагностику устройства для определения его работоспособности. Выполняется для всех подключенных устройств. Формат команды показан в табл. 11.64.

Таблица 11.64. Формат команды EXECUTE DEVICE DIAGNOSTIC

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	Не используется							
1x7h	90h							

Для инициализации команды следует в регистр 1x7h записать код команды 90h. При успешном выполнении регистры будут содержать следующую информацию:

1. В регистр 1x1h будет помещен диагностический код (см. табл. 11.62).
2. В регистры 1x2h—1x6h будет записана сигнатура (см. табл. 11.63).
3. В регистр 1x7h будет записана информация в соответствии с протоколом команды.

11.2.4.4. Команда *FLUSH CACHE*

Эта команда позволяет сбросить кэш на диск. Является обязательной для всех устройств ATA. Для полного завершения операции команде может понадобиться больше полминуты. Формат команды показан в табл. 11.65.

Таблица 11.65. Формат команды FLUSH CACHE

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	E7h							

Для инициализации команды следует в регистр `1x6h` записать номер устройства DEV (0 или 1), а затем в регистр `1x7h` записать код команды `E7h`. При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x6h` будет хранить номер выбранного устройства.
2. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки регистры будут иметь следующий вид:

1. В регистре `1x1h` бит ABRT будет установлен в 1, если устройство не поддерживает команду или произошло аварийное завершение.
2. Регистр `1x3h` будет хранить LBA адрес сектора (0—7), на котором произошла ошибка.
3. Регистр `1x4h` будет хранить LBA адрес сектора (8—15), на котором произошла ошибка.
4. Регистр `1x5h` будет хранить LBA адрес сектора (16—23), на котором произошла ошибка.
5. Регистр `1x6h` будет хранить номер устройства (бит 4) и LBA адрес сектора в битах 0—3 (24—27).
6. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), DRQ = 0 и ERR = 1.

11.2.4.5. Команда *IDENTIFY DEVICE*

Эта команда позволяет получить различную информацию об устройстве. Является обязательной для всех устройств ATA. Формат команды показан в табл. 11.66.

Таблица 11.66. *Формат команды* `IDENTIFY DEVICE`

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	ECh							

Для инициализации команды следует в регистр `1x6h` записать номер устройства DEV (0 или 1), а затем в регистр `1x7h` записать код команды `ECh`. При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x6h` будет хранить номер выбранного устройства.
2. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

После выполнения команды устройство возвращает через регистр данных 256 слов (по 16 битов), описывающих параметры дисковода. Все зарезервированные биты и слова установлены в 0. Формат возвращаемых данных представлен в табл. 11.67.

Таблица 11.67. *Формат данных для команды* `IDENTIFY DEVICE`

Смещение слова	Описание
0	Конфигурация устройства: 15 — тип интерфейса (0 — ATA), 14—8 не используются, 7 — тип устройства (1 — со сменным носителем), 6—3 не используются, 2 — неполные данные (1 — получены не все возможные данные), 1 не используется, 0 зарезервирован
1	Устарел
2	Определяется производителем
3—6	Устарели
7—8	Зарезервированы
9	Не используется
10—19	Серийный номер устройства (содержит 20 ASCII-символов)
20—21	Не используются
22	Устарел
23—26	Номер версии (содержит 8 ASCII-символов)
27—46	Номер модели (содержит 40 ASCII-символов)
47	Биты 15—8 установлены в `80h`, а биты 7—0 определяют максимальное число для команд многоблочного чтения и записи (`01h`—`FFh`)
48	Зарезервирован
49	Поддерживаемые возможности: 15—14 — резерв, 13 — управление таймером для Standby (1 — поддержка в ATA есть, 0 — таймером должно управлять устройство), 12 — резерв, 11 — поддержка сигнала IORDY (0 — есть, 1 — нет), 10 — состояние сигнала IORDY (1 — выключен), 9 — поддержка LBA (1 — есть), 8 — поддержка DMA (1 — есть), 7—0 не используются

Таблица 11.67 (продолжение)

Смещение слова	Описание
50	Поддерживаемые возможности: бит 15 равен 0, бит 14 равен 1, 13—2 — резерв, 1 — устарел, 0 — равен 1 для индикации минимального значения таймера Standby
51—52	Устарели
53	Состав: 15—3 зарезервированы, 2 — допустимость слова 88 (1 — поле со смещением 88 допустимо), 1 — слов 64—70 (1 — допустимы), 0 — устарел
54—58	Устарели
59	Состав: 15—9 — резерв, 8 — настройка многоблочной передачи секторов (1 — допустима), 7—0 — текущее значение числа секторов для многоблочной передачи
60—61	Полное количество секторов, адресуемых пользователем
62	Устарел
63	Состав: 15—11 — резерв, 10 — режим Multiword DMA 2 (1 — выбран), 9 — режим Multiword DMA 1 (1 — выбран), 7—3 — резерв, 2 — поддержка режима Multiword DMA 2 (1 — есть), 1 — поддержка режима Multiword DMA 1 (1 — есть), 0 — поддержка режима Multiword DMA 0 (1 — есть)
64	Состав: 15—8 — резерв, 7—0 — поддержка режимов PIO
65	Минимальное время передачи для режима Multiword DMA в наносекундах
66	Рекомендуемое производителем время передачи для режима Multiword DMA в наносекундах
67	Минимальное время передачи для режима PIO в наносекундах
68	Минимальное время передачи для режима PIO с IORDY в наносекундах
69—79	Необязательные и зарезервированные поля
80	Главный (major) номер версии интерфейса: 15—8 — резерв, 7 — поддержка версии ATA/ATAPI-7 (1 — есть), 6 — поддержка версии ATA/ATAPI-6 (1 — есть), 5 — поддержка версии ATA/ATAPI-5 (1 — есть), 4 — поддержка версии ATA/ATAPI-4 (1 — есть), 3 — поддержка версии ATA/ATAPI-3 (1 — есть), 2—1 устарели, 0 — резерв
81	Дополнительный (minor) номер версии интерфейса
82	Поддержка команд: 15 устарел, 14 — команда `NOP` (1 — есть), 13 — команда `READ BUFFER` (1 — есть), 12 — команда `WRITE BUFFER` (1 — есть), 11 устарел, 10 — защищенная область хоста (1 — есть), 9 — команда `DEVICE RESET` (1 — есть), 8 — прерывание SERVICE (1 — есть),

Таблица 11.67 (продолжение)

Смещение слова	Описание
(прод.)	7 — прерывание при освобождении шины (1 — есть), 6 — упреждение (1 — есть), 5 — кэширование записи (1 — есть), 4 — поддержка пакетных команд (0 — есть), 3 — управление питанием (1 — есть), 2 — поддержка команд для сменных носителей (1 — есть), 1 — поддержка команд секретности (1 — есть), 0 — поддержка команд Smart (1 — есть)
83	Поддержка команд: 15—14 равны 0, 13 — команда FLUSH CACHE EXT (1 — есть), 12 — команда FLUSH CACHE (1 — есть), 11 — набор команд конфигурации (1 — есть), 10 — 48-битная адресация (1 — есть), 9 — автоматическое управление для акустики (1 — есть), 8 — SET MAX (1 — есть), 7 — резерв, 6 — SET FEATURES (1 — есть), 5 — подача питания в режиме Standby (1 — есть), 4 — уведомления для сменных носителей (0 — есть), 3 — расширенное управление питанием (1 — есть), 2 — поддержка команд CFA (1 — есть), 1 — READ/WRITE DMA QUEUED (1 — есть), 0 — DOWNLOAD MICROCODE (1 — есть)
84	Поддержка команд: 15—14 равны 0, 13 — резерв, 12 — ограничение по времени для записи и чтения непрерывно (1 — есть), 11 — ограничение по времени для записи и чтения (1 — есть), 10 — бит URG для команд WRITE STREAM DMA и WRITE STREAM PIO (1 — есть), 9 — бит URG для команд READ STREAM DMA и READ STREAM PIO (1 — есть), 8 — уникальные имена (1 — есть), 7 — команда WRITE DMA QUEUED FUA EXT (1 — есть), 6 — WRITE DMA FUA EXT и WRITE MULTIPLE FUA EXT (1 — есть), 5 — набор свойств регистрации (1 — есть), 4 — набор потоковых свойств (1 — есть), 3 — Media Card (1 — есть), 2 — серийный номер для Media Card (1 — есть), 1 — самотестирование SMART (1 — есть), 0 — отчет об ошибках для SMART (1 — есть)
85	Установка набора команд или свойств: 15 устарел, 14 — команда NOP (1 — включить), 13 — READ BUFFER (1 — включить), 12 — WRITE BUFFER (1 — включить), 11 устарел, 10 — защищенная область хоста (1 — включить), 9 — DEVICE RESET (1 — включить), 8 — прерывание SERVICE (1 — включить), 7 — освобождение шины (1 — включить), 6 — упреждение (1 — включить), 5 — кэширование записи (1 — включить), 4 — упреждение (1 — включить), 3 — управление питанием (1 — включить), 2 — работа со сменными носителями (1 — включить), 1 — набор команд секретности (1 — включить), 0 — набор команд SMART (1 — включить)
86	Установка набора команд или свойств: биты 15—14 — резерв, 13 — команда FLUSH CACHE EXT (1 — включить), 12 — команда FLUSH CACHE (1 — включить), 11 — набор команд конфигурации (1 — включить), 10 — 48-битная адресация (1 — включить), 9 — автоматическое управление для акустики (1 — включить), 8 — SET MAX (1 — включить), 7 — резерв, 6 — SET FEATURES (1 — включить), 5 — подача питания в режиме Standby (1 — включить), 4 — уведомления для сменных носителей (0 — включить), 3 — расширенное управление питанием (1 — включить), 2 — поддержка команд CFA (1 — включить), 1 — READ/WRITE DMA QUEUED (1 — включить), 0 — DOWNLOAD MICROCODE (1 — включить)

Таблица 11.67 (окончание)

Смещение слова	Описание
87	Установка набора команд или свойств: 15 бит равен 0, 14 бит равен 0, 13 — резерв, 12 — ограничение по времени для записи и чтения непрерывно (1 — включить), 11 — ограничение по времени для записи и чтения (1 — включить), 10 — бит URG для команд `WRITE STREAM DMA` и `WRITE STREAM PIO` (1 — включить), 9 — бит URG для команд `READ STREAM DMA` и `READ STREAM PIO` (1 — включить), 8 — уникальные имена (1 — включить), 7 — команда `WRITE DMA QUEUED FUA EXT` (1 — включить), 6 — `WRITE DMA FUA EXT` и `WRITE MULTIPLE FUA EXT` (1 — включить), 5 — набор свойств регистрации (1 — включить), 4 — набор потоковых свойств (1 — включить), 3 — Media Card (1 — включить), 2 — серийный номер для Media Card (1 — включить), 1 — самотестирование SMART (1 — включить), 0 — отчет об ошибках для SMART (1 — включить)
88—254	Необязательные параметры
255	Контрольное слово: 8—15 — контрольная сумма, 0—7 — сигнатура (`A5h`)

11.2.4.6. Команда *IDENTIFY PACKET DEVICE*

Эта команда позволяет получить различную информацию о пакетном устройстве. Является обязательной для всех устройств ATAPI. Формат команды показан в табл. 11.68.

Таблица 11.68. Формат команды `IDENTIFY PACKET DEVICE`

Регистры	Биты							
	7	6	5	4	3	2	1	0
`1x1h`	Не используется							
`1x2h`	Не используется							
`1x3h`	Не используется							
`1x4h`	Не используется							
`1x5h`	Не используется							
`1x6h`	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
`1x7h`	`A1h`							

Для инициализации команды следует в регистр `1x6h` записать номер устройства DEV (0 или 1), а затем в регистр `1x7h` записать код команды `ECh`.

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x6h` будет хранить номер выбранного устройства.
2. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

После выполнения команды устройство возвращает через регистр данных 256 слов (по 16 битов), описывающих параметры дисковода. Все зарезервированные биты и слова будут установлены в 0. Формат возвращаемых данных представлен в табл. 11.69.

Таблица 11.69. *Формат данных для команды* `IDENTIFY PACKET DEVICE`

Смещение слова	Описание
0	Конфигурация устройства: 15—14 — тип интерфейса (`10b` — ATAPI), 13 — резерв, 12—8 — тип устройства (`00h` — устройство прямого доступа, `01h` — устройство последовательного доступа, `02h` — принтер, `03h` — процессор, `04h` — устройство с однократной записью, `05h` — CD-ROM, `06h` — сканер, `1Fh` — неизвестный тип устройства), 7 — устройство поддерживает сменные носители (1 — да), 6—5 — время установки сигнала DRQ (`00b` — через 3 мс после приема команды, `10b` — через 50 мкс после приема команды), 4—3 — резерв, 2 — неполные данные (1 — получены не все возможные данные), 1—0 — поддерживаемый размер пакетной команды (`00b` — 12 байт, `01b` — 16 байт)
1—9	Зарезервированы
10—19	Серийный номер устройства (содержит 20 ASCII-символов)
20—22	Не используются
23—26	Номер версии (содержит 8 ASCII-символов)
27—46	Номер модели (содержит 40 ASCII-символов)
47—48	Зарезервированы
49	Поддерживаемые возможности: 15 — чередование для DMA (1 — есть), 14 — очередь команд (1 — есть), 13 — перекрытие команд (1 — есть), 12 — программный сброс (устарел), 11 — поддержка сигнала IORDY (0 — есть, 1 — нет), 10 — блокировка сигнала IORDY (1 — есть), 9 равен 1, 8 — поддержка DMA (1 — есть), 7—0 не используются
50—52	Зарезервированы
53	Состав: 15—3 зарезервированы, 2 — допустимость слова 88 (1 — поле со смещением 88 допустимо), 1 — слов 64—70 (1 — допустимы), 0 — устарел
54—62	Зарезервированы

Таблица 11.69 (продолжение)

Смещение слова	Описание
63	Состав: 15—11 резерв, 10 — режим Multiword DMA 2 (1 — выбран), 9 — режим Multiword DMA 1 (1 — выбран), 7—3 — резерв, 2 — поддержка режима Multiword DMA 2 (1 — есть), 1 — поддержка режима Multiword DMA 1 (1 — есть), 0 — поддержка режима Multiword DMA 0 (1 — есть)
64	Состав: 15—8 — резерв, 7—0 — поддержка режимов PIO
65	Минимальное время передачи для режима Multiword DMA в наносекундах
66	Рекомендуемое производителем время передачи для режима Multiword DMA в наносекундах
67	Минимальное время передачи для режима PIO в наносекундах
68	Минимальное время передачи для режима PIO с IORDY в наносекундах
69—79	Необязательные и зарезервированные поля
80	Главный (major) номер версии интерфейса: 15—8 — резерв, 7 — поддержка версии ATA/ATAPI-7 (1 — есть), 6 — поддержка версии ATA/ATAPI-6 (1 — есть), 5 — поддержка версии ATA/ATAPI-5 (1 — есть), 4 — поддержка версии ATA/ATAPI-4 (1 — есть), 3 — поддержка версии ATA/ATAPI-3 (1 — есть), 2—1 устарели, 0 — резерв
81	Дополнительный (minor) номер версии интерфейса
82	Поддержка команд: 15 устарел, 14 — команда `NOP` (1 — есть), 13 — команда `READ BUFFER` (1 — есть), 12 — команда `WRITE BUFFER` (1 — есть), 11 устарел, 10 — защищенная область хоста (1 — есть), 9 — команда `DEVICE RESET` (1 — есть), 8 — прерывание SERVICE (1 — есть), 7 — прерывание при освобождении шины (1 — есть), 6 — упреждение (1 — есть), 5 — кэширование записи (1 — есть), 4 — поддержка пакетных команд (0 — есть), 3 — управление питанием (1 — есть), 2 — поддержка команд для сменных носителей (1 — есть), 1 — поддержка команд секретности (1 — есть), 0 — поддержка команд Smart (1 — есть)
83—84	Зарезервированы
85	Установка набора команд или свойств: 15 устарел, 14 — команда `NOP` (1 — включить), 13 — `READ BUFFER` (1 — включить), 12 — `WRITE BUFFER` (1 — включить), 11 устарел, 10 — защищенная область хоста (1 — включить), 9 — `DEVICE RESET` (1 — включить), 8 — прерывание SERVICE (1 — включить), 7 — освобождение шины (1 — включить), 6 — упреждение (1 — включить), 5 — кэширование записи (1 — включить), 4 — упреждение (1 — включить), 3 — управление питанием (1 — включить), 2 — работа со сменными носителями (1 — включить), 1 — набор команд секретности (1 — включить), 0 — набор команд SMART (1 — включить)
86—87	Зарезервированы

Таблица 11.69 (окончание)

Смещение слова	Описание
88	Режимы DMA: 15 — резерв, 14 — выбор Ultra DMA 6 (1 — выбран), 13 — выбор Ultra DMA 5 (1 — выбран), 12 — выбор Ultra DMA 4 (1 — выбран), 11 — выбор Ultra DMA 3 (1 — выбран), 10 — выбор Ultra DMA 2 (1 — выбран), 9 — выбор Ultra DMA 1 (1 — выбран), 8 — выбор Ultra DMA 0 (1 — выбран), 7 — резерв, 6 — поддержка Ultra DMA 6 (1 — есть), 5 — поддержка Ultra DMA 5 (1 — есть), 4 — поддержка Ultra DMA 4 (1 — есть), 3 — поддержка Ultra DMA 3 (1 — есть), 2 — поддержка Ultra DMA 2 (1 — есть), 1 — поддержка Ultra DMA 1 (1 — есть), 0 — поддержка Ultra DMA 0 (1 — есть),
89—255	Зарезервированы

11.2.4.7. Команда *IDLE*

Команда позволяет задавать промежуток времени для перевода устройства в режим простоя (idle). Команда является обязательной для устройств АТА. Формат команды показан в табл. 11.70.

Таблица 11.70. *Формат команды* `IDLE`

Регистры	Биты							
	7	6	5	4	3	2	1	0
`1x1h`	Не используется							
`1x2h`	Значение времени простоя (timevalue)							
`1x3h`	Не используется							
`1x4h`	Не используется							
`1x5h`	Не используется							
`1x6h`	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
`1x7h`	`E3h`							

В регистре `1x2h` задается промежуток времени. Возможные значения показаны в табл. 11.71.

Для инициализации команды в регистр `1x2h` следует записать код времени, в `1x6h` — номер устройства DEV (0 или 1), а затем в регистр `1x7h` записать код команды `E3h`. При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр 1x6h будет хранить номер выбранного устройства.
2. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки регистры будут иметь следующий вид:

1. В регистре 1x1h бит ABRT будет установлен в 1, если устройство не поддерживает команду или произошло аварийное завершение.
2. Регистр 1x6h будет хранить номер устройства (бит 4).
3. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), DRQ = 0 и ERR = 1.

Таблица 11.71. *Значения времени для команды* IDLE

Код времени	Значение времени
00h	Время простоя не используется
01h—F0h	(timevalue * 5) *сек*
F1h—FBh	((timevalue – 240) * 30) мин
FCh	21 мин
FDh	от 8 до 12 часов
FEh	Резерв
FFh	21 мин 15 сек

11.2.4.8. Команда *IDLE IMMEDIATE*

Эта команда позволяет немедленно перевести устройство в режим простоя. Команда является обязательной для устройств ATA. Формат команды показан в табл. 11.72.

Таблица 11.72. *Формат команды* IDLE IMMEDIATE

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	E1h							

Для инициализации команды в регистр `1x6h` следует записать номер устройства DEV (0 или 1), а затем в регистр `1x7h` записать код команды `E1h`. При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x6h` будет хранить номер выбранного устройства.
2. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки регистры будут иметь следующий вид:

1. В регистре `1x1h` бит ABRT будет установлен в 1, если устройство не поддерживает команду или произошло аварийное завершение.
2. Регистр `1x6h` будет хранить номер устройства (бит 4).
3. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), DRQ = 0 и ERR = 1.

Рассмотрим пример для перевода жесткого диска в режим простоя (листинг 11.38).

Листинг 11.38. Перевод жесткого диска в дежурный режим

```
call IsReadATA; ждем, пока порт освободится
mov DX, 1F6h  ; порт 1F6h
mov AL, 0A0h  ; первое устройство на первом канале
out DX, AL    ; записываем значение в порт
inc DX        ; порт 1F7h
mov AL, 0E1h  ; код команды IDLE IMMEDIATE
out DX, AL    ; записываем значение в порт
```

Аналогичный пример для C++ показан в листинге 11.39.

Листинг 11.39. Перевод жесткого диска в дежурный режим в C++

```
// функция для перевода устройства в режим простоя
void SetIdle ()
{
  IsReadyATA (); // ждем, пока порт освободится
  outPort ( 0x1F6, 0xA0, 1); // первое устройство на первом канале
  outPort ( 0x1F7, 0xE5, 1); // код команды CHECK POWER MODE
}
```

В рассматриваемых примерах не обрабатываются ошибки, чтобы максимально упростить принципы использования управляющих команд.

11.2.4.9. Команда *NOP*

Команда позволяет прервать все невыполненные команды, размещенные в очереди, и установить бит аварийного завершения ABRT. Команда является обязательной для устройств ATAPI. Формат команды показан в табл. 11.73.

Таблица 11.73. *Формат команды* NOP

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Дополнительный код							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	00h							

Для инициализации команды в регистр 1x1h следует записать дополнительный код (табл. 11.74), в 1x6h — номер устройства DEV (0 или 1), а затем в регистр 1x7h записать код команды 00h.

Таблица 11.74. *Дополнительный код для команды* NOP

Дополнительный код	Описание
00h (NOP)	Прерывает выполнение очереди команды и возвращает аварийное завершение (ABRT)
01h (NOP Auto Poll)	Не прерывает выполнение команд и возвращает аварийное завершение (ABRT)
02h—FFh (резерв)	Не прерывает выполнение команд и возвращает аварийное завершение (ABRT)

Данная команда всегда возвращает ошибку. В регистре 1x1h бит ABRT устанавливается в 1, а регистр 1x7h имеет вид: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), DRQ = 0 и ERR = 1.

11.2.4.10. Команда *PACKET*

Данная команда позволяет подготовить устройство для последующей передачи пакетной команды. Команда используется только для устройств ATAPI.

Важно помнить, что данную команду всегда нужно вызывать перед применением любой пакетной команды из набора SCSI. Формат команды показан в табл. 11.75.

Таблица 11.75. *Формат команды* PACKET

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется						OVL	DMA
1x2h	Очередь команд					Не используется		
1x3h	Не используется							
1x4h	Количество байтов передачи (0—7)							
1x5h	Количество байтов передачи (8—15)							
1x6h	1	Резерв	1	DEV	Резерв	Резерв	Резерв	Резерв
1x7h	A0h							

В регистре 1x1h следует установить режим передачи в бите DMA (1 — DMA, 0 — PIO) и возможность перекрытия команд в бите OVL (1 — использовать). Если применяется очередь команд, в регистр 1x2h (7—3) следует записать значение тэга для команды (от 0 до 31), иначе этот регистр не используется. В регистры 1x4h и 1x5h следует поместить младший и старший байты для размера блока данных, используемых последующей пакетной командой. К слову, если пакетная команда из набора SCSI не содержит дополнительных параметров и данных, регистры 1x4h и 1x5h не используются.

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр 1x2h будет иметь вид: 0 — установлен в 1, 1 — установлен в 1, 2 — 0, 7—3 — значение от 0 до 31 при поддержке очереди команд.
2. Регистр 1x6h будет хранить номер выбранного устройства.
3. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DMRD = 1 (для DMA), SERV = 1 (при использовании очереди команд), DRQ = 0 и CHK = 0.

В случае ошибки регистры будут иметь следующий вид:

1. Регистр 1x7h: бит ILI зависит от команды, бит EOM зависит от команды, бит ABRT = 1, поле Sense key (7—4) будет содержать код ошибки согласно спецификации SCSI.
2. Регистр 1x2h: C/D = 1, I/O = 1, REL = 0, поле очереди команд (7—3), значение от 0 до 31.

3. Регистр 1x6h будет хранить номер устройства (бит 4).
4. Регистр 1x7h: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), SERV = 1 (если есть перекрытие), DRQ = 0 и CHK = 1.

Например, чтобы подготовить дисковод к приему простой SCSI-команды, не имеющей параметров, можно сделать так, как показано в листинге 11.40.

Листинг 11.40. Подготовка дисковода к работе с пакетными командами

```
call IsReadATA; ждем, пока порт освободится
mov DX, 171h  ; порт 171h
mov AL, 0     ; использовать режим PIO
out DX, AL    ; записываем значение в порт
inc DX        ; порт 172h
mov AL, 0     ; не используем
out DX, AL    ; записываем значение в порт
inc DX        ; порт 173h пропускаем
inc DX        ; порт 174h
mov AL, 0     ; данных для передачи не будет
out DX, AL    ; записываем значение в порт
inc DX        ; порт 175h
mov AL, 0     ; данных для передачи не будет
out DX, AL    ; записываем значение в порт
inc DX        ; порт 176h
mov AL, 0B0h  ; первое устройство на втором канале
out DX, AL    ; записываем значение в порт
inc DX        ; порт 177h
mov AL, 0A0h  ; код команды PACKET
out DX, AL    ; записываем значение в порт
```

Аналогичный пример для C++ показан в листинге 11.41.

Листинг 11.41. Подготовка дисковода к работе с пакетными командами в C++

```
// функция для подготовки устройства к приему пакетной команды
void SetPacket ()
{
   IsReadyATA (); // ждем, пока порт освободится
   outPort ( 0x171, 0x00, 1); // использовать режим PIO
   outPort ( 0x172, 0x00, 1); // не используем
   outPort ( 0x174, 0x00, 1); // данных для передачи не будет
   outPort ( 0x175, 0x00, 1); // данных для передачи не будет
   outPort ( 0x176, 0xB0, 1); // первое устройство на втором канале
   outPort ( 0x177, 0xA0, 1); // код команды PACKET
}
```

11.2.4.11. Команда *READ DMA*

Эта команда позволяет считывать данные с использованием канала DMA. Команда является обязательной для устройств ATA. Формат команды показан в табл. 11.76.

Таблица 11.76. *Формат команды* READ DMA

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Число секторов							
1x3h	Адрес LBA (0—7)							
1x4h	Адрес LBA (8—15)							
1x5h	Адрес LBA (16—24)							
1x6h	1	Резерв	1	DEV	Адрес LBA (24—27)			
1x7h	C8h							

В регистр 1x2h следует записать количество секторов (от 1 до 256), которые будут переданы. Если установить этот регистр в 00h, будет использовано значение в 256 секторов. В регистры 1x3h, 1x4h и 1x5h записывается начальный адрес сектора, с которого начнется передача данных. В регистр 1x5h записывается номер выбранного устройства на канале (бит 4) и старшие 4 разряда адреса сектора (биты 0—3). В регистр 1x7h записывается код команды C8h.

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр 1x6h будет хранить номер выбранного устройства.
2. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

11.2.4.12. Команда *READ MULTIPLE*

Эта команда предназначена для считывания с диска определенного количества секторов. Команда является обязательной для устройств ATA. Формат команды показан в табл. 11.77.

В регистр 1x2h следует записать количество секторов (от 1 до 256), которые будут переданы. Если установить этот регистр в 00h, будет использовано значение 256 секторов. В регистры 1x3h, 1x4h и 1x5h записывается начальный адрес сектора, с которого начнется передача данных. В регистр 1x5h

записываются номер выбранного устройства на канале (бит 4) и старшие 4 разряда адреса сектора (биты 3—0). В регистр `1x7h` записывается код команды `C4h`.

Таблица 11.77. *Формат команды* `READ MULTIPLE`

Регистры	Биты							
	7	6	5	4	3	2	1	0
`1x1h`	Не используется							
`1x2h`	Число секторов							
`1x3h`	Адрес LBA (0—7)							
`1x4h`	Адрес LBA (8—15)							
`1x5h`	Адрес LBA (16—24)							
`1x6h`	1	Резерв	1	DEV	Адрес LBA (24—27)			
`1x7h`	`C4h`							

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x6h` будет хранить номер выбранного устройства.
2. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

11.2.4.13. Команда *READ SECTOR (S)*

Команда позволяет прочитать с диска указанное количество секторов и является обязательной для устройств ATA. Формат команды показан в табл. 11.78.

Таблица 11.78. *Формат команды* `READ SECTOR (S)`

Регистры	Биты							
	7	6	5	4	3	2	1	0
`1x1h`	Не используется							
`1x2h`	Число секторов							
`1x3h`	Адрес LBA (0—7)							
`1x4h`	Адрес LBA (8—15)							
`1x5h`	Адрес LBA (16—24)							
`1x6h`	1	Резерв	1	DEV	Адрес LBA (24—27)			
`1x7h`	`20h`							

В регистр 1x2h следует записать количество секторов (от 1 до 256), которые будут переданы. Если установить этот регистр в 00h, будет использовано значение в 256 секторов. В регистры 1x3h, 1x4h и 1x5h записывается начальный адрес сектора, с которого начнется передача данных. В регистр 1x5h записывается номер выбранного устройства на канале (бит 4) и старшие 4 разряда адреса сектора (биты 3—0). В регистр 1x7h записывается код команды 20h.

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр 1x6h будет хранить номер выбранного устройства.
2. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки команда завершит выполнение. При этом состояние регистров будет таким:

1. Регистр 1x1h: бит NM (1) будет установлен в 1 при отсутствии сменного носителя, бит ABRT (2) будет установлен в 1; бит MCR (3) будет установлен в 1 при наличии сигнала для смены носителя; бит IDNF (4) будет установлен в 1, если указанный адрес не найден; бит MC (5) будет установлен в 1 при попытке смены носителя во время выполнения команды; бит UNC (6) будет установлен в 1 при некорректных данных.
2. Регистры 1x3h, 1x4h и 1x5h будут содержать адрес сектора, при чтении которого произошла ошибка.
3. Регистр 1x6h будет хранить номер устройства (бит 4) и младшие разряды адреса сектора.
4. Регистр 1x7h: ERR = 1, DRQ = 0, DF = 1 (при сбое устройства), DRDY = 1 и BSY = 0.

Рассмотрим пример кода для чтения сектора жесткого диска (листинг 11.42).

Листинг 11.42. Чтение сектора жесткого диска

```
; выделяем буфер для одного сектора 512 байт
data_sector db 512 dup (?)
; код программы
call IsReadATA; ждем, пока порт освободится
mov DX, 1F2h ; порт 1F2h
mov AL, 1    ; читаем один сектор
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F3h
mov AL, 1    ; сектор 1
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F4h
mov AL, 0    ; цилиндр 0
```

```
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F5h
mov AL, 0    ; старший байт цилиндра равен 0
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F6h
mov AL, 0A0h ; первое устройство на первом канале и режим CHS
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F7h
mov AL, 20h  ; команда READ SECTOR
out DX, AL   ; записываем значение в порт
call IsReadATA; ждем, пока порт освободится
mov DX, 1F0h ; порт 1F0h для данных
mov CX, 256  ; количество читаемых слов ( 512 / 2)
mov DI, offset data_sector ; указываем адрес буфера
rep insw     ; читаем все данные
```

Аналогичный пример на C++ будет выглядеть так, как показано в листинге 11.43.

Листинг 11.43. Чтение сектора жесткого диска в C++

```
// пишем функцию для чтения одного сектора
void ReadSector ()
{
  // буфер для данных
   WORD wData[256];
   IsReadyATA (); // ждем, пока порт освободится
   outPort ( 0x1F2, 0x01, 1); // читаем один сектор
   outPort ( 0x1F3, 0x01, 1); // номер сектора
   outPort ( 0x1F4, 0x00, 1); // цилиндр 0
   outPort ( 0x1F5, 0x00, 1); // старший байт цилиндра равен 0
   outPort ( 0x1F6, 0xA0, 1); // первое устройство на первом канале
                              // и режим CHS
   outPort ( 0x1F7, 0x20, 1); // код команды PACKET
   IsReadyATA (); // ждем, пока порт освободится
   // чимтаем данные в буфер
   for ( int i = 0; i < 256; i++)
   {
     inPort ( 0x1F0, ( PDWORD) &( wData[i]), 2);
   }
}
```

Функцию можно сделать универсальной, добавив аргументы для номера сектора, числа секторов и указателя на буфер данных. При этом, естественно, следует изменить код внутри самой функции.

11.2.4.14. Команда *READ VERIFY SECTOR (S)*

Данная команда позволяет читать данные с диска с проверкой, но, в отличии от предыдущей команды, не передает их на компьютер. Команда является обязательной для устройств ATA. Формат команды показан в табл. 11.79.

Таблица 11.79. *Формат команды* READ VERIFY SECTOR (S)

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Число секторов							
1x3h	Адрес LBA (0—7)							
1x4h	Адрес LBA (8—15)							
1x5h	Адрес LBA (16—24)							
1x6h	1	Резерв	1	DEV	Адрес LBA (24—27)			
1x7h	40h							

В регистр 1x2h следует записать количество секторов (от 1 до 256), которые будут переданы. Если установить этот регистр в 00h, будет использовано значение в 256 секторов. В регистры 1x3h, 1x4h и 1x5h записывается начальный адрес сектора, с которого начнется передача данных. В регистр 1x5h записывается номер выбранного устройства на канале (бит 4) и старшие 4 разряда адреса сектора (биты 3—0). В регистр 1x7h записывается код команды 40h.

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр 1x6h будет хранить номер выбранного устройства.
2. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки команда завершит выполнение. При этом состояние регистров будет таким:

1. Регистр 1x1h: бит NM (1) будет установлен в 1 при отсутствии сменного носителя; бит ABRT (2) будет установлен в 1; бит MCR (3) будет установлен в 1 при наличии сигнала для смены носителя; бит IDNF (4) будет установлен в 1, если указанный адрес не найден; бит MC (5) будет установлен в 1 при попытке смены носителя во время выполнения команды; бит UNC (6) будет установлен в 1 при некорректных данных.

2. Регистры 1x3h, 1x4h и 1x5h будут содержать адрес сектора, при чтении которого произошла ошибка.
3. Регистр 1x6h будет хранить номер устройства (бит 4) и младшие разряды адреса сектора.
4. Регистр 1x7h: ERR = 1, DRQ = 0, DF = 1 (при сбое устройства), DRDY = 1 и BSY = 0.

11.2.4.15. Команда *SLEEP*

Команда позволяет перевести устройство в режим "сна" и является обязательной для устройств ATA. Формат команды показан в табл. 11.80.

Таблица 11.80. Формат команды SLEEP

Регистры	Биты							
	7	6	5	4	3	2	1	0
1x1h	Не используется							
1x2h	Не используется							
1x3h	Не используется							
1x4h	Не используется							
1x5h	Не используется							
1x6h	1	Резерв	1	DEV	Не используется			
1x7h	E6h							

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр 1x6h будет хранить номер выбранного устройства.
2. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки регистры будут иметь следующий вид:

1. В регистре 1x1h бит ABRT будет установлен в 1, если устройство не поддерживает команду или произошло аварийное завершение.
2. Регистр 1x6h будет хранить номер устройства (бит 4).
3. Регистр 1x7h будет иметь вид: BSY = 0, DRDY = 1, DF = 1 (при сбое устройства), DRQ = 0 и ERR = 1.

11.2.4.16. Команда *WRITE SECTOR (S)*

Эта команда позволяет записать указанное количество секторов на диск. Команда является обязательной для устройств ATA. Формат команды показан в табл. 11.81.

Таблица 11.81. Формат команды `WRITE SECTOR (S)`

Регистры	Биты							
	7	6	5	4	3	2	1	0
`1x1h`	Не используется							
`1x2h`	Число секторов							
`1x3h`	Адрес LBA (0—7)							
`1x4h`	Адрес LBA (8—15)							
`1x5h`	Адрес LBA (16—24)							
`1x6h`	1	Резерв	1	DEV	Адрес LBA (24—27)			
`1x7h`	`30h`							

В регистр `1x2h` следует записать количество секторов (от 1 до 256), которые будут переданы. Если установить этот регистр в `00h`, будет использовано значение в 256 секторов. В регистры `1x3h`, `1x4h` и `1x5h` записывается начальный адрес сектора, с которого начнется передача данных. В регистр `1x5h` записывается номер выбранного устройства на канале (бит 4) и старшие 4 разряда адреса сектора (биты 3—0). В регистр `1x7h` записывается код команды `30h`.

При успешном выполнении регистры будут содержать следующую информацию:

1. Регистр `1x6h` будет хранить номер выбранного устройства.
2. Регистр `1x7h` будет иметь вид: BSY = 0, DRDY = 1, DF = 0, DRQ = 0 и ERR = 0.

В случае ошибки команда завершит выполнение. При этом состояние регистров будет таким:

1. Регистр `1x1h`: бит NM (1) будет установлен в 1 при отсутствии сменного носителя; бит ABRT (2) будет установлен в 1; бит MCR (3) будет установлен в 1 при наличии сигнала для смены носителя; бит IDNF (4) будет установлен в 1, если указанный адрес не найден; бит MC (5) будет установлен в 1 при попытке смены носителя во время выполнения команды; бит UNC (6) будет установлен в 1 при некорректных данных.
2. Регистры `1x3h`, `1x4h` и `1x5h` будут содержать адрес сектора, при чтении которого произошла ошибка.
3. Регистр `1x6h` будет хранить номер устройства (бит 4) и младшие разряды адреса сектора.
4. Регистр `1x7h`: ERR = 1, DRQ = 0, DF = 1 (при сбое устройства), DRDY = 1 и BSY = 0.

Перед тем как приводить пример работы с этой командой, хочу предупредить читателей о серьезной опасности выполнения кода записи на диск. Пример приводится исключительно в демонстрационных целях и его ни в коем случае не следует повторять. Игнорирование этого замечания может привести к потере информации или выходу из строя жесткого диска! Рассмотрим пример кода для записи сектора жесткого диска (листинг 11.44).

Листинг 11.44. Запись сектора на жесткий диск

```
; выделяем буфер для одного сектора 512 байт
data_sector db 512 dup (0)
; код программы
call IsReadATA; ждем, пока порт освободится
mov DX, 1F2h ; порт 1F2h
mov AL, 1    ; записываем один сектор
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F3h
mov AL, 1    ; сектор 1
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F4h
mov AL, 0    ; цилиндр 0
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F5h
mov AL, 0    ; старший байт цилиндра равен 0
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F6h
mov AL, 0A0h ; первое устройство на первом канале и режим CHS
out DX, AL   ; записываем значение в порт
inc DX       ; порт 1F7h
mov AL, 30h  ; команда WRITE SECTOR
out DX, AL   ; записываем значение в порт
call IsReadATA; ждем, пока порт освободится
mov DX, 1F0h ; порт 1F0h для данных
mov CX, 256  ; количество читаемых слов ( 512 / 2)
mov SI, offset data_sector ; указываем адрес буфера
rep outsw    ; записываем все данные
```

Аналогичный пример на C++ представлен в листинге 11.45.

Листинг 11.45. Запись сектора на жесткий диск в C++

```
// пишем функцию для записи одного сектора
void WriteSector ()
{
   // буфер для данных
   WORD wData[256];
```

```
   memset ( &wData, 0xFF, 256);
   IsReadyATA (); // ждем, пока порт освободится
   outPort ( 0x1F2, 0x01, 1); // записываем один сектор
   outPort ( 0x1F3, 0x01, 1); // номер сектора
   outPort ( 0x1F4, 0x00, 1); // цилиндр 0
   outPort ( 0x1F5, 0x00, 1); // старший байт цилиндра равен 0
   outPort ( 0x1F6, 0xA0, 1); // первое устройство на первом канале
                               // и режим CHS
   outPort ( 0x1F7, 0x30, 1); // код команды WRITE SECTOR
   IsReadyATA (); // ждем, пока порт освободится
   // пишем данные в порт
   for ( int i = 0; i < 256; i++)
   {
     outPort ( 0x1F0, wData[i], 2);
   }
}
```

На этом можно завершить тему программирования портов ввода-вывода. Многое осталось не сказанным, но полученные здесь сведения помогут вам без проблем разобраться со всем остальным. Я же хотел бы познакомить вас с некоторыми способами программирования дисковой подсистемы через интерфейс Win32 API.

11.3. Использование Win32 API

Для работы с дисковой подсистемой используется универсальная функция `DeviceIoControl`. Она является каналом между программой и драйвером устройства. Мы разберем несколько способов ее использования. Для начала попробуем применить эту функцию для открывания лотка CD-ROM (или аналогичного устройства со сменными носителями). Исходный код для выполнения данной задачи представлен в листинге 11.46.

Листинг 11.46. Использование функции `DeviceIoControl` для открывания лотка

```
#include "stdafx.h"
#include <Winioctl.h>
#include <Mmsystem.h>
// значение флага и константы
#define VWIN32_DIOC_DOS_IOCTL 1
#define CF_FLAG 0x0001
// определяем структуру для хранения значений регистров
typedef struct _DIOC_REGISTERS
{
  DWORD reg_EBX;
```

```
  DWORD reg_EDX;
  DWORD reg_ECX;
  DWORD reg_EAX;
  DWORD reg_EDI;
  DWORD reg_ESI;
  DWORD reg_Flags;
} DIOC_REGISTERS, *PDIOC_REGISTERS;
// функция для открытия лотка
void Eject_98ME ( bOpen); // открывает и закрывает лотки
                             // для всех устройств
// объявляем переменные модуля
DIOC_REGISTERS Reg;
// реализуем функцию Eject
void Eject_98ME ( bOpen)
{
   HANDLE hDevice = NULL;
   bool bResult = false;
   DWORD dwResult = 0;
   DWORD dwDrives = 0;
   int nPos = 0, iCH = 0;
   unsigned int uDriveType = 0;
   TCHAR szLetter[8];
    ZeroMemory ( &Reg, sizeof ( Reg));
    // определяем число установленных устройств
    dwDrives = GetLogicalDrives ();
    if( bOpen) // открыть лоток
    {
     while ( dwDrives)
     {
       if ( dwDrive & 1) // если есть диск
        {
          iCH = 0x41 + nPos;
          strcpy ( szLetter, ( const char*) &iCH);
          strcat ( szLetter, ":\\");
          // определяем тип устройства
          uDriveType = GetDriveType ( szLetter);
          // если устройство со сменным носителем
          if ( uDriveType == DRIVE_CDROM)
          {
           // открываем драйвер устройства
           hDevice = CreateFile ( "\\\\.\\vwin32", 0, 0, NULL, 0,
                                  FILE_FLAG_DELETE_ON_CLOSE, NULL);
           if ( hDevice != INVALID_HANDLE_VALUE)
                             // не удалось открыть драйвер
           {
```

```
          // заполняем поля структуры
          Reg.reg_EAX = 0x440D; // номер функции
          Reg.reg_EBX = nPos + 1; // номер устройства
          Reg.reg_ECX = MAKEWORD ( 0x49, 0x08); // код подфункции
          Reg.reg_Flags = 0x0001; // устанавливаем флаг переноса
          // вызываем функцию DeviceIoControl
          bResult = DeviceIoControl ( hDevice, VWIN32_DIOC_DOS_IOCTL,
                                      &Reg, sizeof ( Reg), &Reg,
                                      sizeof ( Reg), & dwResult, 0);
          // проверяем результат операции и флаг CF
          if ( !bResult || ( Reg.reg_Flags & CF_FLAG))
          {
            // закрываем драйвер устройства
            CloseHandle ( hDevice);
            hDevice = NULL;
          }
        }
      // закрываем драйвер устройства
      CloseHandle ( hDevice);
      hDevice = NULL;
      }
    }
   dwDrives >>= 1;
   nPos ++;
   strcpy ( szLetter, "");
  }
 }
else // закрыть лотки
{
 while ( dwDrives)
 {
  if ( dwDrives & 1)
  {
    iCH =  0x41 + nPos;
    strcpy ( szLetter, ( const char*) &iCH);
    strcat( szLetter, ":\\");
    // определяем тип устройства
    uDriveType = GetDriveType ( szLetter);
    if ( uDriveType == DRIVE_CDROM) // если устройство со сменным
                                    // носителем
    {
      // выполняем функцию MCI для закрытия лотка
      mciSendString ( "Set CDAudio Door Closed Wait"), NULL,
                      NULL, NULL);
    }
   }
```

```
      dwDrives >>= 1;
      nPos ++;
      strcpy ( szLetter, "");
     }
   }
}
```

Не забудьте добавить в опции компоновщика ссылку на библиотеку Winmm.lib. Рассмотренная выше функция рассчитана на работу в Windows 95/98/ME. Для использования ее в Windows NT/2000/XP/SR3 придется немного изменить код, согласно листингу 11.47.

Листинг 11.47. Использование функции DeviceIoControl в Windows NT/2000/XP/SR3

```
#include "stdafx.h"
#include <Winioctl.h>
// функция для открытия лотка
void Eject_NT ( bOpen); // открывает и закрывает лотки
                           // для всех устройств
// реализуем функцию Eject
void Eject_NT ( bOpen)
{
   HANDLE hDevice = NULL;
   DWORD dwCode = 0, dwDrives = 0, dwResult = 0;
   TCHAR szLetter[8];
   TCHAR szDeviceName[15];
   int nPos = 0, iCH = 0;
   unsigned int uDriveType = 0;
   if( bOpen) // открыть лоток
     dwCode = IOCTL_STORAGE_EJECT_MEDIA;
   else // закрыть лоток
     dwCode = IOCTL_STORAGE_LOAD_MEDIA;
   // определяем число установленных устройств
   dwDrives = GetLogicalDrives ();
   while ( dwDrives)
   {
    if ( dwDrives & 1) // если есть диск
    {
      iCH = 0x41 + nPos;
      strcpy ( szLetter, ( const char*) & iCH);
      strcat ( szLetter, ":");
      // определяем тип устройства
      uDriveType = GetDriveType ( szLetter);
      if ( uDriveType == DRIVE_CDROM) // если устройство со сменными
                                        // носителями
```

```
      {
        // получаем имя для устройства
        strcpy ( szDeviceName, "\\\\.\\");
        strcat( szDeviceName, szLetter);
        // открываем устройство
        hDevice = CreateFile ( szDeviceName, GENERIC_READ,
                               FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
                               OPEN_EXISTING, 0, NULL);
        if ( hDevice != INVALID_HANDLE_VALUE) // если устройство открыто
        {
          // вызываем функцию DeviceIoControl
          DeviceIoControl ( hDevice, dwCode, NULL, 0, NULL,
                            0, &dwResult, NULL);
          // закрываем устройство
          CloseHandle ( hDevice);
          strcpy ( szDeviceName, "");
        }
      }
    }
    // переходим к следующему
    dwDrives >>= 1;
    nPos ++;
    strcpy ( szLetter, "");
  }
}
```

Представленные вашему вниманию тексты кодов достаточно упрощены для понимания. Теперь поговорим о возможности блокировать лоток устройства, поддерживающего такую возможность. Для выполнения задачи нам потребуется все та же функция `DeviceIoControl`. Чтобы блокировать лоток в операционных системах Windows 95/98/ME, следует написать код, показанный в листинге 11.48.

Листинг 11.48. Блокировка лотка в Windows 95/98/ME

```
#include "stdafx.h"
#include <Winioctl.h>
// значение флага и константы
#define VWIN32_DIOC_DOS_IOCTL 1
#define CF_FLAG 0x0001
// определяем структуры
#pragma pack ( 1)
typedef struct _PARAMBLOCK
{
   BYTE bOperation;
```

```
   BYTE bNumLocks;
} PARAMBLOCK, *PPARAMBLOCK;
#pragma pack ()
typedef struct _DIOC_REGISTERS
{
  DWORD reg_EBX;
  DWORD reg_EDX;
  DWORD reg_ECX;
  DWORD reg_EAX;
  DWORD reg_EDI;
  DWORD reg_ESI;
  DWORD reg_Flags;
} DIOC_REGISTERS, *PDIOC_REGISTERS;
// функция для блокировки лотка
void Lock_98ME ( bLock);
// объявляем переменные модуля
DIOC_REGISTERS Reg;
PARAMBLOCK lockTray;
// реализуем функцию Lock_98ME
void Lock_98ME ( bLock)
{
   HANDLE hDevice = NULL;
   bool bResult = false;
   DWORD dwResult = 0;
   DWORD dwDrives = 0;
   int nPos = 0, iCH = 0;
   unsigned int uDriveType = 0, uFlag = 0;
   TCHAR szLetter[8];
    ZeroMemory ( &Reg, sizeof ( Reg));
    // определяем режим блокировки
    if ( bLock) // блокировать лоток
      uFlag = 0;
    else // разблокировать лоток
      uFlag = 1;
    // определяем число установленных устройств
    dwDrives = GetLogicalDrives ();
    if( bOpen) // открыть лоток
    {
     while ( dwDrives)
     {
       if ( dwDrive & 1) // если есть диск
        {
          iCH = 0x41 + nPos;
          strcpy ( szLetter, ( const char*) &iCH);
          strcat ( szLetter, ":\\");
```

```
          // определяем тип устройства
          uDriveType = GetDriveType ( szLetter);
          // если устройство со сменным носителем
          if ( uDriveType == DRIVE_CDROM)
          {
           // открываем драйвер устройства
           hDevice = CreateFile ( "\\\\.\\vwin32", 0, 0, NULL, 0,
                                   FILE_FLAG_DELETE_ON_CLOSE, NULL);
           if ( hDevice != INVALID_HANDLE_VALUE)
                            // не удалось открыть драйвер
           {
             // заполняем поля структуры
             lockTray.bOperation = uFlag;
             Reg.reg_EAX = 0x440D; // номер функции
             Reg.reg_EBX = nPos + 1; // номер устройства
             Reg.reg_ECX = MAKEWORD ( 0x48, 0x08); // код подфункции
             Reg.reg_EDX = ( DWORD) &lockTray; // указатель на структуру
             Reg.reg_Flags = 0x0001; // устанавливаем флаг переноса
             // вызываем функцию DeviceIoControl
             bResult = DeviceIoControl ( hDevice, VWIN32_DIOC_DOS_IOCTL,
                                         &Reg, sizeof ( Reg), &Reg,
                                         sizeof ( Reg), & dwResult, 0);
             // проверяем результат операции и флаг CF
             if ( !bResult || ( Reg.reg_Flags & CF_FLAG))
             {
               // закрываем драйвер устройства
               CloseHandle ( hDevice);
               hDevice = NULL;
             }
           }
          // закрываем драйвер устройства
          CloseHandle ( hDevice);
          Sleep ( 50); // небольшая задержка
          hDevice = NULL;
          }
       }
      dwDrives >>= 1;
      nPos ++;
      strcpy ( szLetter, "");
    }
}
```

Чтобы создать аналогичную функцию для Windows NT/2000/XP/SR3, следует написать код, показанный в листинге 11.49.

Листинг 11.49. Блокировка лотка в Windows NT/2000/XP/SR3

```
#include "stdafx.h"
#include <Winioctl.h>
// функция для открытия лотка
void Lock_NT ( bLock); // открывает и закрывает лотки
                        // для всех устройств
// реализуем функцию Lock_NT
void Lock_NT ( bLock)
(
   HANDLE hDevice = NULL;
   DWORD dwDrives = 0, dwResult = 0;
   TCHAR szLetter[8];
   TCHAR szDeviceName[15];
   int nPos = 0, iCH = 0;
   unsigned int uDriveType = 0;
   PREVENT_MEDIA_REMOVAL lockTray;
   ZeroMemory ( &lockTray, sizeof ( PREVENT_MEDIA_REMOVAL));
   if( bLock) // блокировать лоток
     lockTray.PreventMediaRemoval = true;
   else // разблокировать лоток
     lockTray.PreventMediaRemoval = false;
   // определяем число установленных устройств
   dwDrives = GetLogicalDrives ();
   while ( dwDrives)
   {
    if ( dwDrives & 1) // если есть диск
    {
      iCH = 0x41 + nPos;
      strcpy ( szLetter, ( const char*) & iCH);
      strcat ( szLetter, ":");
      // определяем тип устройства
      uDriveType = GetDriveType ( szLetter);
      if ( uDriveType == DRIVE_CDROM) // если устройство со сменными
                                      // носителями
      {
        // получаем имя для устройства
        strcpy ( szDeviceName, "\\\\.\\");
        strcat( szDeviceName, szLetter);
        // открываем устройство
        hDevice = CreateFile ( szDeviceName, GENERIC_READ,
                               FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
                               OPEN_EXISTING, 0, NULL);
        if ( hDevice != INVALID_HANDLE_VALUE) // если устройство открыто
        {
```

```
          // вызываем функцию DeviceIoControl
          DeviceIoControl ( hDevice, IOCTL_STORAGE_MEDIA_REMOVAL,
                            (LPVOID) &lockTray,
                            sizeof ( PREVENT_MEDIA_REMOVAL), NULL, 0,
                            &dwResult, NULL);
          // закрываем устройство
          CloseHandle ( hDevice);
          strcpy ( szDeviceName, "");
        }
      }
    }
    // переходим к следующему
    dwDrives >>= 1;
    nPos ++;
    strcpy ( szLetter, "");
  }
}
```

На этом я завершаю тему программирования дисковых устройств в Windows, а читателям советую самостоятельно изучить файл Winioctl.h на предмет других возможностей управления устройствами CD-ROM.

ГЛАВА 12

Пространство шины PCI

Почему, спросите вы, автор называет шину PCI (Peripheral Component Interconnect) пространством. Конечно, в целом шина осталась шиной и выполняет те же функции, что и всегда. Однако современная шина PCI представляет собой не просто общие каналы передачи данных для различных устройств, но и выполняет глобальную управляющую функцию. Она имеет собственное адресное пространство, что позволяет получать доступ ко всем устройствам в системе, независимо от интерфейса. Но самым важным преимуществом шины является возможность гарантированного получения всех доступных портов ввода-вывода и номеров прерываний для последующей работы с устройствами. Не секрет, что немногие системные модули имеют жестко прописанные адреса портов и номера прерываний.

Как узнать, где расположено то или иное устройство? На своем компьютере можно конечно зайти в настройки оборудования и посмотреть, а на чужом придется придумывать что-нибудь другое. Например, попросить пользователя вашей программы ввести нужные данные. Согласитесь, этот способ хорош только для опытных пользователей, не говоря уже о коммерческой несостоятельности такого "интерактивного" проекта. Вот для того, чтобы решить эти и другие проблемы, следует пользоваться возможностями шины PCI. Только в этом случае вам удастся написать приложение, не зависящее от конфигурации оборудования. Исходя из всего сказанного, я хотел бы предложить вам познакомиться с двумя основными способами программирования шины PCI:

1. С помощью функций PCI BIOS.
2. Напрямую через порты ввода-вывода.

12.1. Общие сведения

Современная шина PCI объединяет практически все устройства в компьютере: аудио и видео (AGP и PCI), сетевые и SCSI, южные и северные мос-

ты. Через мосты осуществляется связь с процессором, памятью, кэшем, дисковой подсистемой, мышью и клавиатурой. Шина PCI не просто связывает перечисленные устройства, но также предоставляет удобный интерфейс для доступа и управления ими. Для управления используются так называемые циклы шины с частотой 33 и 66 МГц. Данные частоты являются общими для всех устройств на шине и позволяют синхронизировать работу. Чтобы упорядочить функционирование разнообразного оборудования, шина PCI назначает каждому свой номер. По этому номеру впоследствии определяется тот или иной тип устройства. Кроме того, шина PCI предоставляет общее конфигурационное пространство, которое можно использовать из своей программы для перечисления подключенных устройств и получения различной информации о них (адреса ввода-вывода, идентификационные номера, номера прерываний). Конфигурационное пространство представляет собой шаблон размером 256 байт. Первые 16 байт являются неизменными для любых типов устройств. Остальные могут иметь специфические значения, зависящие от самого устройства. В эти байты записывается различная опознавательная информация, по которой пользователь может определить категорию устройства. Формат пространства шины PCI приведен в табл. 12.1.

Таблица 12.1. *Конфигурационное пространство шины PCI*

Смещение	Биты			
	31–24	23–16	15–8	7–0
00h	Device ID		Vendor ID	
04h	Status		Command	
08h	Class Code			Revision ID
0Ch	BIST	Header Type	Latency Timer	Cache Line Size
10h 14h 18h 1Ch 20h 24h	Base Address Registers			
28h	Cardbus CIS Pointer			
2Ch	Subsystem ID		Subsystem Vendor ID	
30h	Expansion ROM Base Address			
34h	Резерв			
38h	Резерв			
3Ch	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line
40h—FFh	Определяются производителем устройства			

Поскольку программист может получить доступ к пространству шины PCI в режиме чтения и записи, необходимо всегда придерживаться одного правила: для изменения параметров (в том числе отдельных битов) нужно сначала прочитать значение параметра, а затем, изменив желаемые биты, записать их обратно. В обязательном порядке производителями должны поддерживаться следующие поля: Device ID, Vendor ID, Status, Command, Class Code, Revision ID и Header Type. Остальные поля необязательны.

Приведем описание табл. 12.1.

- Vendor ID определяет уникальный идентификатор производителя устройства. Только для чтения. Идентификаторы для наиболее известных в России устройств представлены в табл. 12.2.

Таблица 12.2. *Идентификаторы производителей устройств (Vendor ID)*

Vendor ID	Производитель
0E11h	Compaq
1000h	Symbios Logic (LSI Logic)
1002h	ATI Technologies
1005h	Advance Logic (ADL) Inc
1008h	Epson
100Ah	Phoenix Technologies
100Bh	National Semiconductor
1013h	Cirrus Logic
1014h	IBM
1016h	Fujitsu ICL Personal Systems
1019h	Elitegroup Computer Sys
101Ch	Western Digital
1020h	Hitachi Computer Electronics
1022h	Advanced Micro Devices (AMD)
1023h	Trident Microsystems
1024h	Zenith Data Systems
1025h	Acer Inc
1028h	Dell Computer Corp
1029h	Siemens Nixdorf AG
102Bh	Matrox Graphics Inc

Таблица 12.2 (продолжение)

Vendor ID	Производитель
102Eh	Olivetti Advanced Technology
102Fh	Toshiba America
1033h	NEC Electronics Hong Kong
1037h	Hitachi Micro Systems Inc
1039h	Silicon Integrated Systems (SiS)
103Ah	Seiko Epson Corp
103Ch	Hewlett-Packard Company
1043h	Asustek Computer Inc
1044h	Adaptec
1045h	OPTi Inc
1048h	ELSA GmbH
104Ch	Texas Instruments (TI)
104Dh	Sony Corp
1054h	Hitachi Ltd
1057h	Motorola
105Ah	Promise Technology
1064h	Alcatel CIT
1067h	Mitsubishi Electronics
106Ch	Hyundai Electronics America
1070h	Daewoo Telecom Ltd
1073h	YAMAHA Corp
1076h	Chaintech Computer Co Ltd
1078h	Cyrix Corp
107Ch	LG Electronics
107Fh	Data Technology Corp (DTC)
108Eh	Sun Microsystems
1092h	Diamond Multimedia Systems
1099h	Samsung Electronics Co Ltd
109Dh	Zida Technologies Ltd
10A2h	Quantum Corp

Таблица 12.2 (продолжение)

Vendor ID	Производитель
10A9h	Silicon Graphics
10B7h	3COM Corp, Networking Division
10B9h	Acer Labs Incorporated (ALI)
10C4h	Award Software International Inc
10CAh	Fujitsu Microelectronic
10DEh	Nvidia Corp
10E1h	Tekram Technology Corp Ltd
1102h	Creative Labs
1106h	VIA Technologies Inc
1131h	Philips Semiconductors
115Fh	MAXTOR Corp
117Ah	A-Trend Technology
1180h	Ricoh Co Ltd
11ADh	Lite-On Communications Inc
11BDh	Pinnacle Systems Inc
11CFh	Pioneer Electronic Corp
1240h	Marathon Technologies Corp
125Dh	ESS Technology
126Ah	Lexmark International Inc
1274h	Ensoniq
127Ah	Rockwell Semiconductor Systems
12B9h	3COM Corp, Modem Division
12D2h	STB/Nvidia/SGS Thompson
1414h	Microsoft
142Eh	Vitec Multimedia
1458h	Giga-Byte Technology
1563h	A-Trend Technology Co Ltd
4843h	Hercules Computer Technology Inc
5333h	S3 Inc
8086h	Intel Corporation

Таблица 12.2 *(окончание)*

Vendor ID	Производитель
9004h-9005h	Adaptec
A0A0h	Aopen Inc
E000h	Winbond

- Device ID — идентификатор определенного устройства. Только для чтения. Идентификаторы для наиболее известных в России устройств представлены в табл. 12.3.

Таблица 12.3. *Идентификаторы популярных устройств*

Device ID	Название устройства
00D1h	i740 PCI
7020h	USB Controller
4747h	Rage 3D Pro
5043h	Rage 128 Pro AGP 4x
5048h–5058h	Rage 128
5144h	Radeon DDR
5159h	Radeon VE
5245h	Rage 128 GL PCI
6005h	Crystal CS4281 PCI Audio
0002h	PCI to MCA Bridge (IBM)
0047h	PCI to PCI Bridge (IBM)
7004h	AMD-751 CPU to PCI Bridge (AMD)
7007h	AMD-751 PCI to AGP Bridge (AMD)
7404h	AMD-755 USB Open Host Controller (AMD)
7411h	AMD-766 EIDE Controller (AMD)
5240h	EIDE Controller (Acer)
051Eh	MGA-1164SG Mystique 220 AGP (Matrox)
0520h	MGA-G200B Chipset (Matrox)
0525h	MGA-G400 Chipset (Matrox)

Таблица 12.3 (продолжение)

Device ID	Название устройства
1000h	MGA-G100 Chipset PCI (Matrox)
1525h	Fusion G450 AGP (Matrox)
6057h	MiroVIDEO DC10/DC30 (Miro)
1521h	ALI M1521 Aladdin III CPU to PCI Bridge (Acer)
1531h	ALI M1531 Aladdin IV/IV+ CPU to PCI Bridge (Acer)
0018h	Riva 128
0019h	Riva 128ZX
0020h	Riva TNT
0028h	RIVA TNT2
002Bh	Riva TNT2
002Dh	RIVA TNT2 Model 64
002Fh	Vanta
00A0h	RIVA TNT2 Aladdin
0100h	GeForce 256
0101h	GeForce 256 DDR
0102h	GeForce 256 Ultra
0110h	GeForce2 MX
0112h	GeForce2 MX Ultra
0150h	GeForce2 GTS
0151h	GeForce2 GTS DDR
0152h	GeForce2 Ultra
0200h	GeForce3
0002h	EMU10K1 Audio Chipset (Creative)
7002h	PCI Gameport Joystick (Creative)
0305h	VT8363 KT133 System Controller (VIA)
0680h	VT82C680 Apollo P6 (VIA)
1106h	VT82C570 MV IDE Controller (VIA)
0001h	3Dfx Voodoo
0002h	3Dfx Voodoo2
0003h	3Dfx Voodoo Banshee

Таблица 12.3 *(окончание)*

Device ID	Название устройства
0005h	3Dfx Voodoo3
5880h	5880 AudioPCI
8813h	S3 Trio64
8814h	S3 Trio64UV+
8900h	S3 Trio64V2/DX
8A10h	S3 ViRGE/GX2
8A20h	S3 Savage3D
8A22h	S3 Savage4
8A26h	S3 ProSavage
9102h	S3 Savage 2000
0122h	82437FX 430FX (Intel)
1222h	82092AA EIDE Controller (Intel)
1239h	82371FB 430FX PCI EIDE Controller (Intel)
2412h	82801AA 8xx Chipset USB Controllers (Intel)
2416h	82801AA 8xx Chipset AC'97 PCI Modem (Intel)
2418h	82801AA 8xx Chipset Hub to PCI Bridge (Intel)
2421h	82801AB 8xx Chipset IDE Controller (Intel)
2422h	82801AB 8xx Chipset USB Controller (Intel)
2441h	82801BA Bus Master IDE Controller (Intel)
244Ah	82801BAM (ICH2) UltraDMA/100 IDE Controller (Intel)
7010h	82371SB PIIX3 EIDE Controller (Intel)
7020h	82371SB PIIX3 USB Controller (Intel)
7111h	82371AB/EB/MB PIIX4 EIDE Controller (Intel)
7112h	82371AB/EB/MB PIIX4 USB Controller (Intel)
7113h	82371AB/EB/MB PIIX4 Power Management Controller (Intel)

- Command открывает доступ к командному регистру, который позволяет осуществлять грубую регулировку циклов шины PCI. Для чтения и записи. Формат регистра показан в табл. 12.4. Следует помнить, что не каждое устройство поддерживает данное поле.

Таблица 12.4. *Формат командного регистра (Command)*

Биты	Описание
0	Управление возможностью устройства принимать и посылать данные в пространстве PCI (1 — включить, 0 — выключить)
1	Управление возможностью устройства работать с памятью (1 — включить, 0 — выключить)
2	Управление возможностью устройства работать в качестве ведущего на шине PCI (1 — включить, 0 — выключить)
3	Поддержка устройством специальных циклов шины (1 — включить контроль, 0 — игнорировать)
4	Управление сигналом для записи в память (1 — включить, 0 — выключить)
5	Управление доступом к регистрам палитры для VGA-совместимых и других графических устройств (1 — разрешить доступ к палитре, 0 — запретить доступ к палитре)
6	Поддержка устройством ошибок четности (1 — включить, 0 — выключить)
7	Резерв
8	Резерв
9	Возможность транзакций для устройства (1 — включить, 0 — выключить)
10—15	Резерв

- Status является необязательным полем. Позволяет получить информацию о состоянии шины PCI. Для чтения и записи. Формат регистра показан в табл. 12.5.

Таблица 12.5. *Формат регистра состояния (Status)*

Биты	Описание
0—4	Резерв
5	Только для чтения. Указывает на поддержку устройством частоты шины 66 МГц (1 — поддерживает 66 МГц, 0 — только 33 МГц)
6	Только для чтения. Указывает на поддержку устройством пользовательских настроек конфигурации (1 — да, 0 — нет)
7	Только для чтения. Указывает на поддержку устройством транзакций (1 — да, 0 — нет)
8	Резерв
9—10	Определяют типы синхронизации для устройства

Таблица 12.5 (окончание)

Биты	Описание
11	Бит устанавливается в 1 при аварийном завершении транзакции для целевого устройства
12	Бит устанавливается в 1 ведущим устройством при аварийном завершении транзакции
13	Бит устанавливается в 1 ведущим устройством при аварийном завершении транзакции
14	Резерв
15	Бит устанавливается в 1 при обнаружении устройством ошибок четности

- Revision ID определяет номер версии устройства в дополнении к Device ID. Только для чтения.
- Class Code определяет класс устройства. Только для чтения. Формат этого поля представлен в табл. 12.6. Код базового класса и подкласса позволяют идентифицировать тип устройства. Возможные значения кодов для базового класса показаны в табл. 12.7. Значения кодов подкласса и интерфейса перечислены в табл. 12.8. Формат байта интерфейса для контроллеров IDE представлен в табл. 12.9.

Таблица 12.6. Формат поля Class Code

Смещение	0Bh	0Ah	09h
Описание	Код базового класса	Код подкласса	Интерфейс

Таблица 12.7. Коды базового класса

Код базового класса	Описание
00h	Устройство появилось до введения кодов класса
01h	Контроллер устройства хранения большой емкости
02h	Сетевой контроллер
03h	Контроллер дисплея
04h	Мультимедийное устройство
05h	Контроллер памяти
06h	Устройство моста
07h	Обычный контроллер связи

Таблица 12.7 (окончание)

Код базового класса	Описание
08h	Системная периферия
09h	Устройство ввода
0Ah	Стыковочный узел
0Bh	Процессор
0Ch	Контроллер последовательной шины
0Dh—FEh	Резерв
FFh	Устройство не подходит ни под один класс

***Таблица 12.8.** Коды подкласса и интерфейса*

Базовый класс	Подкласс	Интерфейс	Описание
00h	00h	00h	Все выпущенные ранее устройства, кроме VGA-совместимых
	01h	00h	VGA-совместимые устройства
01h	00h	00h	Контроллер SCSI
	01h	xxh	Контроллер IDE
	02h	00h	Контроллер флоппи-дисковода
	03h	00h	Контроллер IPI
	04h	00h	Контроллер RAID
	05h	20h	Контроллер ATA с одиночным DMA
		30h	Контроллер ATA с цепочкой DMA
	06h	00h	Контроллер SATA
		01h	Контроллер SATA с интерфейсом AHCI 1.0
	07h	00h	Контроллер SAS
	80h	00h	Контроллер для другого устройства хранения
02h	00h	00h	Контроллер Ethernet
	01h	00h	Контроллер Token Ring
	02h	00h	Контроллер FDDI
	03h	00h	Контроллер ATM
	80h	00h	Сетевой контроллер другого типа

Таблица 12.8 *(продолжение)*

Базовый класс	Подкласс	Интерфейс	Описание
03h	00h	00h	VGA-совместимый контроллер
		01h	8514–совместимый контроллер
	01h	00h	Контроллер XGA
	02h	00h	Контроллер 3D
	80h	00h	Контроллер дисплея другого типа
04h	00h	00h	Устройство для работы с видео
	01h	00h	Устройство для работы с аудио
	80h	00h	Другое мультимедийное устройство
05h	00h	00h	Контроллер оперативной памяти
	01h	00h	Контроллер флэш-памяти
	80h	00h	Другой тип контроллера памяти
06h	00h	00h	Мост хоста
	01h	00h	Мост ISA
	02h	00h	Мост EISA
	03h	00h	Мост MCA
	04h	00h	Мост PCI-to-PCI
	05h	00h	Мост PCMCIA
	06h	00h	Мост NuBus
	07h	00h	Мост CardBus
	80h	00h	Другой тип моста
07h	00h	00h	Контроллер последовательного порта (XT-совместимый)
		01h	16450-совместимый контроллер последовательного порта
		02h	16450-совместимый контроллер последовательного порта
	01h	00h	Контроллер параллельного порта
		01h	Контроллер двунаправленного параллельного порта
		02h	Контроллер параллельного порта ECP 1.X
	80h	00h	Контроллер другого устройства связи

Таблица 12.8 *(окончание)*

Базовый класс	Подкласс	Интерфейс	Описание
08h	00h	00h	Контроллер прерываний 8259
		01h	Контроллер прерываний ISA
		02h	Контроллер прерываний EISA
	01h	00h	Контроллер DMA 8237
		01h	Контроллер DMA ISA
		02h	Контроллер DMA EISA
	02h	00h	Контроллер системного таймера 8254
		01h	Контроллер системного таймера ISA
		02h	Контроллер системного таймера EISA
	03h	00h	Контроллер часов реального времени
		01h	Контроллер часов реального времени ISA
	80h	00h	Другой тип контроллера
09h	00h	00h	Контроллер клавиатуры
	01h	00h	Контроллер дигитайзера
	02h	00h	Контроллер мыши
	80h	00h	Другой тип контроллера ввода
0Ah	00h	00h	Контроллер стыковочного узла
	80h	00h	Другой тип контроллера
0Bh	00h	00h	386
	01h	00h	486
	02h	00h	Pentium
	10h	00h	Alpha
	20h	00h	Power PC
	40h	00h	Сопроцессор
0Ch	00h	00h	Контроллер IEEE 1394
	01h	00h	ACCESS
	02h	00h	SSA
	03h	00h	USB
	04h	00h	Fibre Channel

Таблица 12.9. *Формат байта интерфейса для контроллеров IDE*

Биты	7	6	5	4	3	2	1	0
Описание	Master	0	0	0	Режим	Режим	Режим	Режим

Приведем краткий комментарий к табл. 12.9.

- Бит 0 определяет режим работы для первого канала контроллера. Если бит установлен в 1, канал работает в режиме PCI, иначе — в режиме совместимости.
- Бит 1 указывает на поддержку первичным каналом фиксированного режима выполнения операций (0 — режим установлен, 1 — поддерживаются оба режима).
- Бит 2 определяет режим работы для вторичного канала контроллера. Если бит установлен в 1, канал работает в режиме PCI, иначе — в режиме совместимости.
- Бит 3 указывает поддержку вторичным каналом фиксированного режима выполнения операций (0 — режим установлен, 1 — поддерживаются оба режима).
- Биты 4—6 зарезервированы и должны быть установлены в 0.
- Бит 7 определяет поддержку выбора ведущего устройства на канале.

- Cache Line Size является необязательным. Данное поле доступно, если устройство поддерживает запись в память. Для чтения и записи.
- Latency Timer является необязательным. Определяет время ожидания для ведущего устройства на шине PCI. Для чтения и записи.
- Header Type определяет информацию со смещения `10h`. Бит 7 указывает на многофункциональное устройство (1). Биты 0—6 идентифицируют формат дальнейшей информации (`00h` — формат согласно данной таблице, `01h` — формат определяется спецификацией моста PCI-to-PCI). Только для чтения.
- BIST является необязательным полем. Управляет режимом самотестирования для устройства. Если устройство не поддерживает самодиагностику, значение поля будет равно 0. Бит 7 указывает на поддержку режима самотестирования (1 — есть). Запись в бит 6 единицы позволит выполнить самотестирование указанного устройства. Биты 0—3 определяют результат операции (0 — ошибка). Для чтения и записи.
- Base Address Registers содержит все доступные адреса ввода-вывода для работы с устройством. Для чтения и записи.
- Cardbus CIS Pointer используется для специфических устройств связи. Подробнее можно узнать из спецификации устройств PCMCIA. Только для чтения.

- Interrupt Line позволяет получить номер выделенной для устройства линии прерывания. Для чтения и записи.
- Min_Gnt и Max_Lat определяют время ожидания и параметры его настройки. Время ожидания измеряется в микросекундах. Только для чтения.

Остальные поля в табл. 12.1 необязательны и могут отличаться для различных устройств.

12.2. Использование функций PCI BIOS

Функции PCI BIOS предназначены для работы во всех режимах X86: защищенном и реальном, 16-ти и 32-разрядном. Для 16-разрядных программ доступ к функциям возможен через прерывание `int 1Ah` (физический адрес `000FFE6Eh`). Доступ в 32-разрядных программах осуществляется через вызов точки входа в защищенный режим. Разработчики рекомендуют выделять для стека не менее 1024 байт.

Перед началом работы всегда следует убедиться в наличии PCI BIOS. В реальном режиме для этого применяется специальная функция, в защищенном режиме следует использовать физический адрес. Данные о PCI BIOS размером 16 байт могут находиться в диапазоне адресов `0E0000h`—`0FFFFFh`. Формат данных показан в табл. 12.10.

Таблица 12.10. Формат структуры PCI BIOS

Смещение поля	Размер поля	Описание
00h	4 байта	Сигнатура в виде ASCII-символов "_32_"
04h	4 байта	32-разрядный физический адрес точки входа
08h	1 байт	Младший номер версии (00h, 01h или 02h)
09h	1 байт	Размер структуры (16 байт)
0Ah	1 байт	Контрольная сумма (сумма всех полей должна равняться 0)
0Bh	5 байтов	Зарезервированы и должны быть установлены в 0

Следует просканировать диапазон адресов `0E0000h`—`0FFFFFh`, пока не будет найдена сигнатура PCI BIOS ("_32_").

А теперь рассмотрим функции PCI BIOS.

12.2.1. Функция *B101h*

Эта функция позволяет определить наличие в системе поддержки PCI BIOS.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 01h.
3. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код состояния (00h, если PCI BIOS имеется). В регистр EDX будет помещена сигнатура "PCI" (в DL 'P', в DH 'C', далее 'I' и в старший байт пробел). В регистр AL будет записан код аппаратного механизма. Имеются два таких механизма: первый (01h) и второй (02h). В BH будет сохранено значение старшего номера версии, а в BL — младшего номера. Например, для версии 2.10 в BH будет записано значение 02h, а в BL — 10h. В регистр CL будет помещен номер последней шины в системе. Кроме того, в случае ошибки флаг CF будет установлен, иначе сброшен. Рассмотрим пример для проверки наличия PCI BIOS в системе (листинг 12.1).

Листинг 12.1. Проверка поддержки набора функций PCI BIOS

```
mov AH, 0B1h ; код функции B1h
mov AL, 01h  ; код подфункции 01h
int 1Ah      ; вызываем прерывание
cmp AH, 00h  ; состояние выполнения функции
jne NO_PCI   ; поддержка PCI BIOS в системе отсутствует
cmp EDX, 20494350h; проверяем сигнатуру "PCI " с пробелом
jne NO_PCI   ; что угодно, только не PCI BIOS
; определяем количество шин и сохраняем на всякий случай
mov byte ptr NUM_BUS, CL
```

12.2.2. Функция *B102h*

Функция позволяет найти определенное устройство на шине. Для этого следует знать Vendor ID и Device ID желаемого устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 02h.
3. В регистр CX записывается идентификатор устройства (Device ID). Диапазон значений лежит от 0000h до FFFFh.

4. В регистр DX записывается идентификатор производителя (Vendor ID). Диапазон значений лежит от 0000h до FFFEh.
5. В регистр SI нужно записать номер индекса. Если вы хотите найти более одного устройства, следует установить индекс в соответствующее значение. Начальное значение равно 0.
6. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (табл. 12.11). В регистр BH будет записан номер шины (от 0 до 255). В BL будет помещен номер устройства (биты 3—7) и номер функции (0—2). В случае ошибки флаг CF будет установлен, иначе сброшен.

***Таблица 12.11.** Возвращаемые значения функций PCI BIOS*

Код значения	Описание
00h	Операция успешно завершена
81h	Указанная функция не поддерживается
83h	Недопустимый идентификатор производителя (Vendor ID)
86h	Устройство не найдено
87h	Недопустимый номер регистра
88h	Ошибка выполнения
89h	Недостаточный размер буфера

Рассмотрим пример поиска видеоадаптера GeForce3, представленный в листинге 12.2.

Листинг 12.2. Поиск видеоадаптера на шине PCI

```
mov AH, 0B1h ; код функции B1h
mov AL, 02h  ; код подфункции 02h
mov CX, 0200h; Device ID для GeForce3
mov DX, 10DEh; Vendor ID для Nvidia
mov SI, 0    ; индекс 0
int 1Ah      ; вызываем прерывание
cmp AH, 86h  ; проверяем результат операции
jne NO_FIND  ; устройство не найдено
```

12.2.3. Функция *B103h*

Функция позволяет найти устройства указанного класса. Можно использовать данную функцию, если вы не знаете точных идентификаторов устройства и производителя.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 03h.
3. В регистр ECX следует записать значение кода класса (в младшие три байта).
4. В регистр SI нужно записать номер индекса. Если вы хотите найти более одного устройства, следует установить индекс в соответствующее значение. Начальное значение равно 0.
5. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11). В регистр BH будет записан номер шины (от 0 до 255). В регистр BL будет помещен номер устройства (биты 3—7) и номер функции (0—2). В случае ошибки флаг CF будет установлен, иначе сброшен. Рассмотрим пример для поиска основного моста хоста, приведенный в листинге 12.3.

Листинг 12.3. Поиск системной микросхемы моста

```
mov AH, 0B1h ; код функции B1h
mov AL, 03h  ; код подфункции 03h
mov ECX, 060000h; код класса
mov SI, 0    ; индекс 0
int 1Ah      ; вызываем прерывание
cmp AH, 86h  ; проверяем результат операции
jne NO_FIND  ; устройство не найдено
; сохраняем номер шины и устройства
mov byte ptr NUM_BUS, BH
mov byte ptr NUM_DEV, BL
```

12.2.4. Функция *B106h*

Данная функция позволяет сгенерировать специальный цикл для шины PCI. Сгенерированный цикл будет использован на указанном номере шины.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 06h.

3. В регистр BH следует записать номер шины (0—255).
4. В EDX необходимо поместить данные для специального цикла шины.
5. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11). В случае ошибки флаг CF будет установлен, иначе сброшен.

12.2.5. Функция *B108h*

Функция позволяет прочитать байт из конфигурационного пространства указанного устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 08h.
3. В регистр BH следует записать номер шины (0—255).
4. В регистр BL нужно записать номер устройства (биты 3—7) и номер функции (биты 0—2).
5. В регистр DI следует указать номер байта (регистра). Возможные значения лежат в диапазоне от 0 до 255.
6. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11), а в регистр CL— значение прочитанного байта. В случае ошибки флаг CF будет установлен, иначе сброшен. В листинге 12.4 показан пример кода для получения типа заголовка (Header Type).

Листинг 12.4. Получение описателя устройства

```
; сначала следует найти устройство
mov AH, 0B1h ; код функции B1h
mov AL, 03h  ; код подфункции 03h
mov ECX, 060000h; код класса
mov SI, 0    ; индекс 0
int 1Ah      ; вызываем прерывание
cmp AH, 86h  ; проверяем результат операции
jne NO_FIND  ; устройство не найдено
mov AH, 0B1h ; код функции B1h
mov AL, 08h  ; код подфункции 08h
```

```
mov DI, 0Eh  ; смещение для Header Type
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
```

12.2.6. Функция *B109h*

Эта функция позволяет прочитать слово (два байта) из конфигурационного пространства указанного устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 09h.
3. В регистр BH следует записать номер шины (0—255).
4. В регистр BL нужно записать номер устройства (биты 3—7) и номер функции (биты 0—2).
5. В DI следует указать смещение слова (16-битного регистра). Возможные значения получаются следующим образом: 0, 2, 4, 6, 8, ..., 254.
6. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11), а в регистр CX — значение прочитанного слова. В случае ошибки флаг CF будет установлен, иначе сброшен. Чтобы прочитать идентификатор производителя, выполните код из листинга 12.5.

Листинг 12.5. Получение идентификатора производителя

```
VENDOR_ID dw ?
; сначала следует найти устройство
mov AH, 0B1h ; код функции B1h
mov AL, 03h  ; код подфункции 03h
mov ECX, 030000h; код класса контроллера видеоадаптера
mov SI, 0    ; индекс 0
int 1Ah      ; вызываем прерывание
cmp AH, 86h  ; проверяем результат операции
jne NO_FIND  ; устройство не найдено
mov AH, 0B1h ; код функции B1h
mov AL, 09h  ; код подфункции 09h
mov DI, 0    ; смещение для Vendor ID
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение идентификатора
mov word ptr VENDOR_ID, CX
```

12.2.7. Функция *B10Ah*

Функция позволяет прочитать двойное слово (четыре байта) из конфигурационного пространства указанного устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 0Ah.
3. В регистр BH следует записать номер шины (0—255).
4. В регистр BL нужно записать номер устройства (биты 3—7) и номер функции (биты 0—2).
5. В DI следует указать смещение слова (32-битного регистра). Возможные значения получаются следующим образом: 0, 4, 8, 19, 32, ..., 252.
6. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11), а в регистр ECX — значение прочитанного слова. В случае ошибки флаг CF будет установлен, иначе сброшен. Чтобы прочитать базовые адреса, можно использовать следующий код, показанный в листинге 12.6.

Листинг 12.6. Получение базового адреса устройства

```
; сначала следует найти устройство
mov AH, 0B1h ; код функции B1h
mov AL, 03h  ; код подфункции 03h
mov ECX, 010000h; код класса контроллера SCSI
mov SI, 0    ; индекс 0
int 1Ah      ; вызываем прерывание
cmp AH, 86h  ; проверяем результат операции
jne NO_FIND  ; устройство не найдено
; получаем базовый адрес 1
mov AH, 0B1h ; код функции B1h
mov AL, 0Ah  ; код подфункции 0Ah
mov DI, 10h  ; смещение адреса
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение адреса 1
mov dword ptr base_addr_1, ECX
; получаем базовый адрес 2
mov AH, 0B1h ; код функции B1h
mov AL, 0Ah  ; код подфункции 0Ah
```

```
mov DI, 14h  ; смещение адреса
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение адреса 2
mov dword ptr base_addr_2, ECX
; получаем базовый адрес 3
mov AH, 0B1h ; код функции B1h
mov AL, 0Ah  ; код подфункции 0Ah
mov DI, 18h  ; смещение адреса
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение адреса 3
mov dword ptr base_addr_3, ECX
; получаем базовый адрес 4
mov AH, 0B1h ; код функции B1h
mov AL, 0Ah  ; код подфункции 0Ah
mov DI, 1Ch  ; смещение адреса
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение адреса 4
mov dword ptr base_addr_4, ECX
; получаем базовый адрес 5
mov AH, 0B1h ; код функции B1h
mov AL, 0Ah  ; код подфункции 0Ah
mov DI, 20h  ; смещение адреса
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение адреса 5
mov dword ptr base_addr_5, ECX
; получаем базовый адрес 6
mov AH, 0B1h ; код функции B1h
mov AL, 0Ah  ; код подфункции 0Ah
mov DI, 24h  ; смещение адреса
int 1Ah      ; вызываем прерывание
jc  ERROR_HND; произошла ошибка
; сохраняем значение адреса 6
mov dword ptr base_addr_6, ECX
```

12.2.8. Функция *B10Bh*

Данная функция позволяет записать один байт в конфигурационное пространство указанного устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 0Bh.
3. В регистр BH следует записать номер шины (0—255).
4. В регистр BL нужно записать номер устройства (биты 3—7) и номер функции (биты 0—2).
5. В регистр DI следует указать номер байта (регистра). Возможные значения лежат в диапазоне от 0 до 255.
6. В регистр CL следует записать значение байта.
7. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11). В случае ошибки флаг CF будет установлен, иначе сброшен.

12.2.9. Функция *B10Ch*

Функция позволяет записать одно слово (два байта) в конфигурационное пространство указанного устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 09h.
3. В регистр BH следует записать номер шины (0—255).
4. В регистр BL нужно записать номер устройства (биты 3—7) и номер функции (биты 0—2).
5. В регистр DI следует указать смещение слова (16-битного регистра). Возможные значения получаются следующим образом: 0, 2, 4, 6, 8, ..., 254.
6. В регистр CX следует записать значение слова.
7. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11). В случае ошибки флаг CF будет установлен, иначе сброшен.

12.2.10. Функция *B10Dh*

Данная функция позволяет записать двойное слово (четыре байта) в конфигурационное пространство указанного устройства.

Использование:

1. В регистр AH следует поместить код функции B1h.
2. В регистр AL следует поместить код подфункции 0Dh.
3. В регистр BH следует записать номер шины (0—255).
4. В регистр BL нужно записать номер устройства (биты 3—7) и номер функции (биты 0—2).
5. В регистр DI следует указать смещение слова (32-битного регистра). Возможные значения получаются следующим образом: 0, 4, 8, 19, 32, ..., 252.
6. В регистр ECX следует записать значение двойного слова.
7. Вызвать прерывание int 1Ah.

Выход:

После выполнения функции в регистр AH будет помещен код результата (см. табл. 12.11). В случае ошибки флаг CF будет установлен, иначе сброшен.

Чтобы использовать функции записи PCI BIOS, необходимо иметь документацию на программируемое устройство.

На этом мы завершим описание функций BIOS и перейдем к непосредственному программированию портов ввода-вывода.

12.3. Использование портов

Представленный здесь материал основывается на устройствах фирмы Intel. Любую дополнительную информацию по данной теме можно свободно найти на сайте этого производителя (**www.intel.com**). Как правило, для доступа к шине PCI используются адреса 0CF8h—0CFFh. За определенными адресами закреплены аппаратные регистры, позволяющие получить доступ к разнообразным устройствам на шине PCI. Рассмотрим их подробнее.

12.3.1. Регистр конфигурации адреса

Данный регистр расположен по адресу 0CF8h. Этот 32-разрядный регистр доступен для чтения и записи. Он позволяет определить конфигурационные параметры устройства на шине PCI, а также номер регистра для последующего доступа к устройству. Формат этого регистра показан в табл. 12.12. По умолчанию значение регистра равно 00000000h.

Таблица 12.12. Формат регистра конфигурации адреса

Бит	Описание
0—1	Резерв
2—7	Определяют значение регистра для доступа к устройству

Таблица 12.12 (окончание)

Бит	Описание
8—10	Номер функции для многофункционального устройства
11—15	Номер устройства на шине PCI
16—23	Номер шины PCI
24—30	Резерв
31	Установка этого бита в 1 разрешает доступ к конфигурационному пространству шины PCI, иначе доступ заблокирован (бит равен 0)

12.3.2. Регистр конфигурации данных

Регистр расположен по адресу `0CFCh`. Этот 32-разрядный регистр доступен для чтения и записи. Он предназначен для чтения или записи данных в конфигурационное пространство шины PCI. Для обмена данными используются все биты (0—31).

С помощью этих двух регистров можно получить доступ к любому устройству на шине. Для этого в регистр адреса следует записать параметры устройства и после этого можно читать и писать данные через регистр данных. Рассмотрим пример кода для сканирования шины PCI, представленный в листинге 12.7.

Листинг 12.7. Сканирование шины PCI

```
#include "stdafx.h"
// определяем значения смещений для пространства шины PCI
#define VENDOR_ID           0x00
#define DEVICE_ID           0x02
#define CODE                0x04
#define STATUS              0x06
#define REVISION_ID         0x08
#define INTERFACE           0x09
#define SUBCLASS            0x0A
#define CLASSCODE           0x0B
#define CACHE_LINE_SIZE     0x0C
#define LATENCY_TIMER        0x0D
#define HEADER_TYPE         0x0E
#define BIST                0x0F
#define BASE_ADDRESS_0      0x10
#define BASE_ADDRESS_1      0x14
#define BASE_ADDRESS_2      0x18
```

```
#define BASE_ADDRESS_3     0x1C
#define BASE_ADDRESS_4     0x20
#define BASE_ADDRESS_5     0x24
#define CARDBUS_POINTER    0x28
#define SUBVEN_ID          0x2C
#define SUBSYSTEM_ID        0x2E
#define ROM_BASE_ADDRESS    0x30
#define INTERRUPT_LINE      0x3C
#define INTERRUPT_PIN        0x3D
#define MIN_GNT             0x3E
#define MAX_LAT             0x3F
// функция для сканирования шины PCI
void ScanPCI ( int iBusPCI);
// определение номера слота для устройства
int GetDeviceSlot ( int iDevice, int iFunction);
// получаем конфигурационные параметры для устройства
DWORD GetDevice ( int iBusPCI, int iSlot, int iAddress);
// реализуем наши функции
int GetDeviceSlot ( int iDevice, int iFunction)
{
   return ( ( ( ( iDevice) & 0x1f) << 3) | ( ( iFunction) & 0x07));
}
DWORD GetDevice ( int iBusPCI, int iSlot, int iAddress)
{
   return ( 0x80000000L | ( ( iBus & 0xff) << 16) | ( iSlot << 8) |
          ( iAddress & ~3));
}
// единственный аргумент функции определяет номер шины ( от 0 до 255)
void ScanPCI ( int iBusPCI)
{
   DWORD dwResult = 0;
   BYTE buffer[256];
   // создаем двойной цикл для перебора всех устройств
   for ( int iDevice = 0; iDevice < 32; iDevice++)
   {
      for ( int iFunction = 0; iFunction < 8; iFunction++)
      {
        memset ( &buffer, 0, 256);
        // вычисляем номер очередного слота
        int iSlot = GetNumSlot ( iDevice, iFunction);
        // проверяем поле Vendor ID для определения наличия устройства
        DWORD dwVendorID = 0;
        // получаем конфигурацию устройства
        dwResult = GetDevice ( iBusPCI, iSlot, VENDOR_ID);
```

```
        // пишем в адресный порт параметры устройства
        outPort ( 0xCF8, dwResult, 4);
        dwResult = 0;
        // читаем из порта данных идентификатор производителя
        inPort ( 0xCFC, &dwResult, 4);
        // если полученное  значение равно 0 или 0xFFFFFFFF,
        // выходим из вложенного цикла и продолжаем поиск
        if (  dwVendorID == 0x00000000 || dwVendorID == 0xFFFFFFFF)
            break;
        // если устройство присутствует, читаем его параметры
        // из конфигурационного пространства шины PCI
        for ( int j = 1; j < 256; j++)
        {
          // получаем конфигурацию устройства
          dwResult = GetDevice ( iBusPCI, iSlot, j);
          // пишем в адресный порт параметры устройства
          outPort ( 0xCF8, dwResult, 4);
          // получаем из порта очередной байт
          inPort ( 0xCFC + ( j&0x03), &dwResult, 1);
          // сохраняем полученный байт в буфер
          buffer[j] = dwResult;
          // здесь мы можем извлечь нужные данные из буфера и сохранить
          // их для последующего использования
          // например, получим номер прерывания для устройства
          // переменная uINT определена где-то ранее
          uINT = buffer[INTERRUPT_LINE];
          // поле Header Type
          uHeader = buffer[HEADER_TYPE];
          // поле Revision ID
          uRevID = buffer[REVISION_ID];
          // поле Device ID
          dwDeviceID = ( ( WORD) ( ( ( BYTE) ( buffer[PCI_DEVICE_ID]))
  | ( ( ( WORD) ( ( BYTE) ( buffer[PCI_DEVICE_ID] + 1))) << 8)));
          // поле Code
          dwCode = ( ( WORD) ( ( ( BYTE) ( buffer[CODE]))
          | ( ( ( WORD) ( ( BYTE) ( buffer[CODE] + 1))) << 8)));
          // поле Subsystem Vendor ID
          dwSubVendorID = ( ( WORD) ( ( ( BYTE) ( buffer[SUBVEN_ID]))
     | ( ( ( WORD) ( ( BYTE) ( buffer[SUBVEN_ID] + 1))) << 8)));
          // базовый адрес 0
          WORD low = 0, high = 0;
          // получаем младшее слово адреса
          low = ( ( WORD) ( ( ( BYTE) ( buffer[BASE_ADDRESS_0]))
 | ( ( ( WORD) ( ( BYTE) ( buffer[BASE_ADDRESS_0] + 1))) << 8)));
```

```
            // получаем старшее слово адреса
           high = ( ( WORD) ( ( ( BYTE) ( buffer[BASE_ADDRESS_0] + 2))
 | ( ( ( WORD) ( ( BYTE) ( buffer[BASE_ADDRESS_0] + 3))) << 8)));
           // вычисляем полный адрес
           dwBaseAddress_0 = ( ( LONG) ( ( ( WORD) ( low)) |
                       ( ( ( DWORD) ( ( WORD) ( high))) << 16)));
        }
      }
   }
}
// пример вызова функции ScanPCI для первой шины
ScanPCI ( 0);
// пример вызова функции ScanPCI для второй шины
ScanPCI ( 1);
```

Итак, наша функция `ScanPCI` имеет всего один аргумент, в качестве которого используется номер шины PCI. Из примера видно, что, работая всего с двумя портами 0CF8h и 0CFCh, мы получили возможность доступа ко всем устройствам компьютера. Кроме общих сведений о каждом устройстве, нам удалось определить базовые адреса ввода-вывода и номер прерывания. Однако следует заметить, что далеко не все устройства возвращают номер прерывания. Это связано с тем, что наряду с привычными модулями (видеоконтроллер, IDE или звуковой контроллер) возвращаются данные о специфических устройствах, например о мостах, USB-контроллерах и др., использующих совершенно отличные принципы работы.

Фирмы Intel и Via на своих сайтах предоставляют самую свежую информацию о новых устройствах и наборах микросхем (chipset). В этой документации содержится подробная информация о доступе и управлении устройствами посредством конфигурационного пространства шины PCI. Поскольку каждый новый набор микросхем имеет свои особенности, рекомендую читателям постоянно следить за этой информацией.

Давайте в качестве примера рассмотрим конфигурационное пространство шины PCI для современного контроллера IDE фирмы Intel. Формат контроллера (256 байт) представлен в табл. 12.13.

Таблица 12.13. *Контроллер IDE фирмы Intel*

Смещение	Описание
00h—01h	Vendor ID
02h—03h	Device ID
04h—05h	Командный регистр

Таблица 12.13 *(продолжение)*

Смещение	Описание
06h—07h	Status
08h	Revision ID
09h	Интерфейс
0Ah	Код подкласса
0Bh	Код базового класса
0Ch	Резерв
0Dh	Latency Timer
0Eh	Header Type
0Fh	Резерв
10h—13h	Базовый адрес для первого канала
14h—17h	Базовый адрес для первого канала
18h—1Bh	Базовый адрес для второго канала
1Ch—1Fh	Базовый адрес для второго канала
20h—23h	Адрес регистра для шины IDE
24h—27h	Расширения
28h—2Bh	Резерв
2Ch—2Dh	Subsystem Vendor ID
2Eh—2Fh	Subsystem ID
24h—3Bh	Резерв
3Ch	Interrupt Line
3Dh	Interrupt Pin
3Fh	Резерв
40h—41h	Синхронизация для первого канала
42h—43h	Синхронизация для второго канала
44h	Синхронизация для ведомого устройства на первом и втором каналах
45h—47h	Резерв
48h	Управляющий регистр Ultra DMA
49h	Резерв
4Ah—4Bh	Регистр синхронизации Ultra DMA

Таблица 12.13 (окончание)

Смещение	Описание
4Ch—53h	Резерв
54h—55h	Конфигурация ввода-вывода
56h—F7h	Резерв
F8h—FBh	Идентификатор производителя
FCh—FFh	Резерв

Приведем краткое описание таблицы.

- 00h—01h — идентификатор производителя. Только для чтения. Всегда будет равен значению 8086h.
- 02h—03h — идентификатор изделия. Только для чтения. Возможны следующие значения:
  - 2411h — контроллер ATA (82801AA);
  - 2421h — контроллер ATA (82801AB);
  - 244Ah — контроллер ATA для мобильных ПК (82801BA);
  - 244Bh — контроллер ATA для высокопроизводительных ПК (82801BA);
  - 248Ah — контроллер ATA для мобильных ПК (82801CA);
  - 248Bh — контроллер ATA для высокопроизводительных ПК (82801CA);
  - 24CAh — контроллер ATA для мобильных ПК (82801DB);
  - 24CBh — контроллер ATA для высокопроизводительных ПК (82801CA);
  - 24DBh — контроллер ATA для высокопроизводительных ПК (82801EB).
- 04h—05h определяют управляющий регистр. Для чтения и записи. Формат регистра показан в табл. 12.14.

***Таблица 12.14.** Формат управляющего регистра*

Бит	Описание
0	Управление операциями ввода-вывода (1 — разрешить, 0 — запретить)
1	Управление доступом к памяти (1 — разрешить, 0 — запретить)
2	Управление шиной (1 — включить, 0 — выключить)
3—15	Зарезервированы и равны 0

- `06h—07h` определяют регистр состояния. Доступен для чтения и записи.
- `08h` — номер версии устройства. Только для чтения.
- `09h` определяет интерфейсный регистр. Для чтения и записи. Формат регистра показан в табл. 12.15.

***Таблица 12.15.** Формат интерфейсного регистра*

Бит	Описание
0	Выбор режима для первичного канала (1 — использовать совместимый, 0 — использовать "родной" PCI)
1	Поддержка режима на первичном канале (1 — использовать совместимый и "родной", 0 — только совместимый)
2	Выбор режима для вторичного канала (1 — использовать совместимый, 0 — использовать "родной" PCI)
3	Поддержка режима на вторичном канале (1 — использовать совместимый и "родной", 0 — только совместимый)
4—6	Если бит 7 равен 1, используется "родной" режим шины PCI, иначе — совместимый
7	Управляет значением битов 4—6

- `0Ah` определяет код подкласса для устройства. Только для чтения.
- `0Bh` определяет код базового класса для устройства. Только для чтения.
- `0Dh` определяет время ожидания для ведущего устройства на шине PCI. Доступен для чтения и записи.
- `0Eh` определяет многофункциональное устройство. Только для чтения.
- `10h—13h`, `14h—17h`, `18h—1Bh`, `1Ch—1Fh` и `20h—23h` определяют базовые адреса каналов. Доступны для чтения и записи. Формат регистров показан в табл. 12.16.

***Таблица 12.16.** Формат регистра базового адреса*

Бит	Описание
0	Тип ресурса (должен быть равен 1), только для чтения
1—3	Зарезервированы и должны быть установлены в `000h`
4—31	Значение базового адреса ввода-вывода

- 24h—27h определяют базовый адрес ввода-вывода в памяти. Доступны для чтения и записи. Формат регистров показан в табл. 12.17.

Таблица 12.17. *Формат регистра адреса, расположенного в памяти*

Бит	Описание
0	Тип ресурса (должен быть равен 0), только для чтения
1—2	32-разрядный тип отображения в памяти (должен быть равен 0), только для чтения
3	0
4—31	Значение базового адреса ввода-вывода в памяти

- 2Ch—2Dh определяют дополнительный идентификатор производителя, который доступен для чтения и однократной записи (для повторной записи придется перезагрузить компьютер). По умолчанию значение этого регистра равно 0.
- 2Eh—2Fh определяют идентификатор подсистемы. Доступен для чтения и однократной записи (для повторной записи придется перезагрузить компьютер). По умолчанию значение этого регистра равно 0.
- 3Ch определяет номер выделенного прерывания. Доступен для чтения и однократной записи (для повторной записи придется перезагрузить компьютер). По умолчанию значение этого регистра равно 0.
- 3Dh определяет дополнительные опции линии прерывания для контроллера. Только для чтения. Используется в стандартном режиме работы шины PCI. По умолчанию значение этого регистра равно 0.
- 40h—41h и 42h—43h управляют режимом синхронизации контроллера. Доступны для чтения и записи. Формат регистра показан в табл. 12.18.

Таблица 12.18. *Формат регистра синхронизации контроллера*

Бит	Описание
0	Использование ускоренной синхронизации для первого канала (1 — включить, 0 — выключить)
1	Использование сигнала IORDY для первого канала (1 — включить, 0 — выключить)
2	Использование упреждения для первого канала (1 — включить, 0 — выключить)
3	Использование синхронизации с DMA для первого канала (1 — включить, 0 — выключить)

Таблица 12.18 (окончание)

Бит	Описание
4	Использование ускоренной синхронизации для второго канала (1 — включить, 0 — выключить)
5	Использование сигнала IORDY для второго канала (1 — включить, 0 — выключить)
6	Использование упреждения для второго канала (1 — включить, 0 — выключить)
7	Использование синхронизации с DMA для второго канала (1 — включить, 0 — выключить)
8—9	Время восстановления (00h — 4 такта, 01h — 3 такта, 10h — 2 такта, 11h — 1 такт)
10—11	Зарезервированы и должны быть установлены в 00h
12—13	Частота сигнала IORDY (00h — 5 тактов, 01h — 4 такта, 10h — 3 такта, 11h — 2 такта)
14	Управление регистром синхронизации для ведомого устройства (1 — включить, 0 — выключить)
15	Использование декодирования для IDE (1 — включить, 0 — выключить)

- 44h управляет параметрами синхронизации для обоих каналов. Доступен для чтения и записи. Формат регистра показан в табл. 12.19.

Таблица 12.19. *Формат регистра 44h*

Бит	Описание
0—1	Время восстановления для первого ведомого канала (00h — 4 такта, 01h — 3 такта, 10h — 2 такта, 11h — 1 такт)
2—3	Частота сигнала IORDY для первого ведомого канала (00h — 5 тактов, 01h — 4 такта, 10h — 3 такта, 11h — 2 такта)
4—5	Время восстановления для второго ведомого канала (00h — 4 такта, 01h — 3 такта, 10h — 2 такта, 11h — 1 такт)
6—7	Частота сигнала IORDY для второго ведомого канала (00h — 5 тактов, 01h — 4 такта, 10h — 3 такта, 11h — 2 такта)

- 48h определяет параметры синхронизации для режима Ultra DMA. Доступен для чтения и записи. Формат регистра показан в табл. 12.20.

Таблица 12.20. *Формат регистра* `48h`

Бит	Описание
0	Управление режимом Ultra DMA для первого устройства на первом канале (1 — включить, 0 — выключить)
1	Управление режимом Ultra DMA для второго устройства на первом канале (1 — включить, 0 — выключить)
2	Управление режимом Ultra DMA для первого устройства на втором канале (1 — включить, 0 — выключить)
3	Управление режимом Ultra DMA для второго устройства на втором канале (1 — включить, 0 — выключить)
4—7	Зарезервированы и должны быть установлены в `0000h`

- `4Ah—4Bh` определяют дополнительные параметры синхронизации для режима Ultra DMA. Доступны для чтения и записи.
- `54h—55h` определяют конфигурацию контроллера. Доступны для чтения и записи. Формат регистра показан в табл. 12.21.

Таблица 12.21. *Формат регистра конфигурации контроллера*

Бит	Описание
0	Частота для первого диска на первом канале (1 — 66 МГц, 0 — 33 МГц)
1	Частота для второго диска на первом канале (1 — 66 МГц, 0 — 33 МГц)
2	Частота для первого диска на втором канале (1 — 66 МГц, 0 — 33 МГц)
3	Частота для второго диска на втором канале (1 — 66 МГц, 0 — 33 МГц)
4	Тип используемого кабеля для первого диска на первом канале (1 — 80 жил, 0 — 40 жил)
5	Тип используемого кабеля для второго диска на первом канале (1 — 80 жил, 0 — 40 жил)
6	Тип используемого кабеля для первого диска на втором канале (1 — 80 жил, 0 — 40 жил)
7	Тип используемого кабеля для второго диска на втором канале (1 — 80 жил, 0 — 40 жил)
8—9	Зарезервированы и должны быть установлены в `00h`
10	Эффективность передачи данных для режима PIO (1 — включить, 0 — выключить)
11	Зарезервирован и должен быть установлен в `0h`

Таблица 12.21 (окончание)

Бит	Описание
12	Повышение частоты для первого диска на первом канале (1 — 100 МГц, 0 — 33 или 66 МГц)
13	Повышение частоты для второго диска на первом канале (1 — 100 МГц, 0 — 33 или 66 МГц)
14	Повышение частоты для первого диска на втором канале (1 — 100 МГц, 0 — 33 или 66 МГц)
15	Повышение частоты для второго диска на втором канале (1 — 100 МГц, 0 — 33 или 66 МГц)

- `F8h`—`FBh` определяют дополнительный идентификатор производителя. Только для чтения.

Как видите, структура описания контроллера IDE полностью оговаривает все необходимые для работы с дисками параметры. Вам достаточно найти данное устройство на шине PCI (используя функцию `ScanPCI`) и сохранить базовые адреса и требуемую информацию. После этого можно спокойно работать с диском посредством набора команд ATA/ATAPI.

ГЛАВА 13

Контроллер DMA

Контроллер прямого доступа к памяти (DMA — Direct Memory Access) предназначен для обмена данными между оперативной памятью и устройством без помощи центрального процессора. Это значит, что во время операций чтения или записи с использованием каналов DMA процессор свободен и может обрабатывать другие данные, что ощутимо увеличивает общую производительность компьютера.

Контроллер DMA реализован на основе микросхемы 8237 (8237A). Она содержит четыре независимых канала. Число каналов может быть увеличено за счет каскадного подключения дополнительных микросхем 8237. Как правило, в компьютере имеется два контроллера: первый 8-разрядный (поддерживает каналы 0, 1, 2 и 3) и 16-разрядный (поддерживает каналы 4, 5, 6 и 7). Первый позволяет адресацию до 1 Мбайт памяти, а второй — до 16 Мбайт. Кроме того, первый канал поддерживает адресуемые блоки размером 64 Кбайт, а второй — 128 Кбайт, поэтому при передаче данных следует контролировать границы адресуемых участков. Контроллер DMA поддерживает четыре режима работы:

1. Одиночная передача данных.
2. Передача данных блоками.
3. Передача данных по внешнему запросу.
4. Каскадный режим работы.

Первый канал DMA (0) используется для регенерации обновления памяти, и его не рекомендуется программировать. Второй канал (1) поддерживает работу с контроллером гибких дисков. Третий канал DMA (2) отведен для параллельного порта принтера (ECP). Четвертый выполняет роль связующего со вторым контроллером DMA с помощью метода каскадирования. Для доступа к контроллеру применяются порты, перечисленные в табл. 13.1.

Таблица 13.1. Список портов для работы с контроллером DMA

Номер порта	Описание
00h	Регистр данных канала 0
01h	Регистр числа обработанных байтов канала 0
02h	Регистр данных канала 1
03h	Регистр числа обработанных байтов канала 1
04h	Регистр данных канала 2
05h	Регистр числа обработанных байтов канала 2
06h	Регистр данных канала 3
07h	Регистр числа обработанных байтов канала 3
08h	Управляющий регистр (запись) и регистр состояния (чтение) для DMA-1
09h	Регистр запроса для DMA-1
0Ah	Регистр маски для DMA-1
0Bh	Регистр режима работы DMA-1
0Ch	Регистр сброса триггера DMA-1
0Dh	Регистр сброса контроллера DMA-1
0Eh	Регистр сброса маскирующих битов DMA-1
0Fh	Регистр установки масок для всех каналов DMA-1
81h	Регистр адреса страницы канала 2
82h	Регистр адреса страницы канала 3
83h	Регистр адреса страницы канала 1
87h	Регистр адреса страницы канала 0
89h	Регистр адреса страницы канала 6
8Ah	Регистр адреса страницы канала 5
8Bh	Регистр адреса страницы канала 7
8Fh	Регистр обновления памяти
C0h	Регистр данных канала 4
C2h	Регистр числа обработанных байтов канала 4
C4h	Регистр данных канала 5
C6h	Регистр числа обработанных байтов канала 5
C8h	Регистр данных канала 6

Таблица 13.1 (окончание)

Номер порта	Описание
CAh	Регистр числа обработанных байтов канала 6
CCh	Регистр данных канала 7
CEh	Регистр числа обработанных байтов канала 7
D0h	Управляющий регистр (запись) и регистр состояния (чтение) для DMA-2
D2h	Регистр запроса для DMA-2
D4h	Регистр маски для DMA-2
D6h	Регистр режима работы DMA-2
D8h	Регистр сброса триггера DMA-2
DAh	Регистр сброса контроллера DMA-2
DCh	Регистр сброса маскирующих битов DMA-2
DEh	Регистр установки масок для всех каналов DMA-2

Передача данных через порты 00h—07h происходит в два этапа: первые 8 бит (0—7) и вторые 8 бит (8—15). Связано это с тем, что регистры 8-разрядные. Кроме того, эти порты доступны для чтения и записи. В режиме чтения они возвращают младший (00h, 02h, 04h и 06h) и старший (01h, 03h, 05h и 07h) байты для текущего адреса в соответствующем канале DMA-1.

Порты C0h—CEh доступны для чтения и записи. В режиме чтения они возвращают младший (C0h, C4h, C8h и CCh) и старший (C2h, C6h, CAh и CEh) байты для текущего адреса в соответствующем канале DMA-2.

Порты 08h (DMA-1) и D0h (DMA-2) в режиме записи используются как управляющие, а в режиме чтения возвращают текущее состояние. Размер обоих регистров составляет 8 бит. Они позволяют настроить работу всех четырех каналов. Регистры состояния также имеют размер 8 бит и помогают получить состояние каналов контроллера. Формат регистра управления показан в табл. 13.2, регистра состояния — в табл. 13.3.

***Таблица 13.2.** Формат управляющего регистра для DMA-1 и DMA-2*

Бит	Описание
0	Режим память-память (1 — включен, 0 — выключен)
1	Удержание канала 0 (1 — включено, 0 — выключено). Если бит 0 равен 0, не используется
2	Управление контроллером (1 — выключен, 0 — включен)

Таблица 13.2 (окончание)

Бит	Описание
3	Режим сжатия по времени (1 — включен, 0 — выключен). Если бит 0 равен 1, не используется
4	Управление приоритетом (1 — с чередованием, 0 — фиксированный)
5	Цикл записи (1 — расширенный, 0 — с запаздыванием). Если бит 3 равен 1, не используется
6	Сигнал DREQ (1 — выключен, 0 — включен)
7	Сигнал DACK (1 — выключен, 0 — включен)

Таблица 13.3. *Формат регистра состояния для DMA-1 и DMA-2*

Бит	Описание
0	Завершена работа DMA режима для канала 0 (4), если значение равно 1
1	Завершена работа DMA режима для канала 1 (5), если значение равно 1
2	Завершена работа DMA режима для канала 2 (6), если значение равно 1
3	Завершена работа DMA режима для канала 3 (7), если значение равно 1
4	Получен запрос для канала 0 (4), если значение равно 1
5	Получен запрос для канала 1 (5), если значение равно 1
6	Получен запрос для канала 2 (6), если значение равно 1
7	Получен запрос для канала 3 (7), если значение равно 1

Порты `09h` (DMA-1) и `D2h` (DMA-2) используются только в режиме записи и позволяют установить сигнал запроса на указанном канале. Размер обоих регистров составляет 4 бита. Формат регистра запроса показан в табл. 13.4.

Таблица 13.4. *Формат регистра запроса для DMA-1 и DMA-2*

Бит	Описание
0—1	Выбор канала для сигнала запроса (`00b` — канал 0 или 4, `01b` — канал 1 или 5, `10b` — канал 2 или 6, `11b` — канал 3 или 7)
2	Управление сигналом запроса (1 — установить, 0 — сбросить)
3—7	Игнорируются

Порты 0Ah (DMA-1) и D4h (DMA-2) используются только в режиме записи и позволяют установить бит маски на указанном канале для блокировки сигнала DREQ. Размер обоих регистров составляет 4 бита. Формат регистра маски показан в табл. 13.5.

***Таблица 13.5.** Формат регистра маски для DMA-1 и DMA-2*

Бит	Описание
0—1	Выбор канала для сигнала запроса (00b — канал 0 или 4, 01b — канал 1 или 5, 10b — канал 2 или 6, 11b — канал 3 или 7)
2	Управление битом маски (1 — установить, 0 — сбросить)
3—7	Игнорируются

Порты 0Bh (DMA-1) и D6h (DMA-2) используются только в режиме записи и позволяют настроить режим работы контроллера для указанного канала. Номер канала устанавливается в самом регистре (биты 0 и 1). Размер обоих регистров составляет 6 бит. Формат регистра режима показан в табл. 13.6.

***Таблица 13.6.** Формат регистра режима для DMA-1 и DMA-2*

Бит	Описание
0—1	Выбор канала для сигнала запроса (00b — канал 0 или 4, 01b — канал 1 или 5, 10b — канал 2 или 6, 11b — канал 3 или 7)
2—3	Режим передачи данных (00b — с проверкой, 01b — запись, 10b — чтение, 11b — недопустимый). Если биты 6 и 7 установлены в 1, данное поле не используется
4	Автоинициализация (1 — включить, 0 — выключить)
5	Изменение значения адреса (1 — уменьшение, 0 — увеличение)
6—7	Выбор режима работы (00b — передача по запросу, 01b — одиночная передача, 10b — передача блоками, 11b — каскадный режим)

Порты 0Ch (DMA-1) и D8h (DMA-2) используются только в режиме записи и позволяют включить использование портов 00h—07h для работы с 16-разрядными значениями. Вначале нужно будет прочитать младший байт, а затем старший.

Порты 0Dh (DMA-1) и DAh (DMA-2) позволяют выполнить сброс контроллера DMA. Для этого следует записать в эти порты любое значение.

Порты 0Eh (DMA-1) и DCh (DMA-2) позволяют выполнить сброс битов маски для всех каналов. Для этого следует записать в эти порты любое значение.

Порты 0Fh (DMA-1) и DEh (DMA-2) позволяют выполнить установки битов маски для всех каналов. Формат регистра установки маски показан в табл. 13.7.

Таблица 13.7. *Формат регистра установки бита маски для всех каналов*

Бит	Описание
0	Бит маски для канала 0 (4) установлен, если значение равно 1, иначе сброшен
1	Бит маски для канала 1 (5) установлен, если значение равно 1, иначе сброшен
2	Бит маски для канала 2 (6) установлен, если значение равно 1, иначе сброшен
3	Бит маски для канала 3 (7) установлен, если значение равно 1, иначе сброшен
4—7	Зарезервированы и должны быть установлены в 0

Вот и все базовые сведения о контроллере DMA. Рассмотрим пример, позволяющий выполнить сброс контроллера DMA-2 (листинг 13.1).

Листинг 13.1. Сброс контроллера DMA

```
ResetDMA2 proc
mov AL, 01h  ; сюда можно записать любое значение
out 0DAh, AL ; сбрасываем контроллер DMA-2
ret
ResetDMA2endp
```

Аналогичный пример для C++ представлен в листинге 13.2.

Листинг 13.2. Сброс контроллера DMA в C++

```
// пишем функцию для сброса контроллера DMA
void ResetDMA ( unsigned int uDMA)
{
   if (  ( uDMA == 0) || ( uDMA > 2) return;
   if ( uDMA == 1) // DMA-1
     outPort ( 0x0D, 0x01, 1); // выполняем сброс
   else // DMA-2
     outPort ( 0xDA, 0x01, 1); // выполняем сброс
}
// сбрасываем контроллер DMA-1
ResetDMA ( 1);
```

Рассмотрим еще один пример для определения доступности канала 3 на контроллере DMA-1, представленный в листинге 13.3.

Листинг 13.3. Проверка готовности канала DMA

```
IsReadyDMA_3 proc
xor AL, AL   ; обнуляем регистр
in  AL, 08h  ; читаем байт статуса
test AL, 80h ; проверяем бит 7
jnz DMA3_PROC; переходим к дальнейшим действиям
ret
IsReadyDMA_3 endp
```

Проверку готовности канала DMA в C++ можно выполнить так, как показано в листинге 13.4.

Листинг 13.4. Проверка готовности канала контроллера DMA-1 в C++

```
bool IsReadyDMA_1 ( unsigned int uChannel)
{
   DWORD dwResult = 0;
   if ( uChannel > 3) return;
   // получаем байт состояния
   inport ( 0x08, &dwResult, 1); // DMA-1
   // проверяем указанный канал
   switch ( uChannel)
    {
    case 0:
      if ( ( dwResult & 0x10) == 0x01) return true;
     break;
    case 1:
      if ( ( dwResult & 0x20) == 0x01) return true;
     break;
    case 2:
      if ( ( dwResult & 0x40) == 0x01) return true;
     break;
    case 3:
      if ( ( dwResult & 0x80) == 0x01) return true;
     break;
    }
   return false;
}
```

Вот и все, что мне хотелось рассказать о контроллере прямого доступа к памяти. Несомненно, его использование существенно увеличивает скорость обмена данными между устройством и компьютером, но следует помнить, что далеко не все современные устройства поддерживают такой режим работы. Кроме того, перед использованием контроллера необходимо убедиться в том, что выбранный канал не занят другим устройством и готов к работе.

ГЛАВА 14

Контроллер прерываний

Любой компьютер имеет в своем составе разнообразные внешние и внутренние устройства. Каждое из них так или иначе управляется центральным процессором, который в свою очередь должен успевать обрабатывать все поступающие данные. Поскольку процессор не может одновременно работать более чем с один устройством, пришлось придумывать какие-то способы для разделения обработки данных, поступающих от конкретных устройств в отдельно взятой временной фазе. Вполне работоспособным показал себя метод циклического опроса каждого устройства с последующей обработкой данных, однако в результате получилась система с очень низкой производительностью, поскольку большая часть процессорного времени тратилась на банальное повторение одних и тех же операций (опрос, опознавание и последующий обмен данными). Наиболее приемлемым выходом из такой ситуации было бы использование процессора только тогда, когда он нужен определенному устройству. В результате решения этой задачи появилось новое устройство, называемое контроллером прерываний (например, 8259 фирмы Intel). Общий принцип его работы достаточно прост: когда возникает необходимость в использовании процессора, каждое устройство посылает сигнал запроса в контроллер прерываний. В контроллере полученный запрос "рассматривается" и, в зависимости от назначенного устройству приоритета, посылается на выполнение центральному процессору. Процессор, получив запрос от контроллера прерываний, запоминает свое текущее состояние и переключается на выполнение запрошенной операции, по выполнении которой возвращается к решению сохраненной задачи. Такой режим работы называется прерыванием и позволяет достаточно эффективно обслуживать все имеющиеся в системе устройства.

Контроллер прерываний состоит из двух микросхем 8259 (в настоящее время они интегрируются в общий набор микросхем), подключенных по каскадной схеме. Каждая из них имеет 8 линий прерываний, закрепленных за определенным устройством. Первая микросхема обслуживает прерывания

от IRQ0 до IRQ7, а вторая — от IRQ8 до IRQ15. Все прерывания имеют свой уровень приоритета обслуживания процессором. Самый высокий приоритет назначен прерыванию IRQ0, а самый низкий — IRQ15. В табл. 14.1 показано стандартное назначение каждого прерывания. В скобках после номера прерывания указан приоритет в числовом выражении.

Таблица 14.1. *Список аппаратных прерываний*

Номер прерывания	Назначенное устройство
IRQ0 (0)	Таймер
IRQ1 (1)	Клавиатура
IRQ2	Каскадное подключение второго контроллера
IRQ3 (10)	Последовательный порт COM2
IRQ4 (11)	Последовательный порт COM1
IRQ5 (12)	Параллельный порт LPT2
IRQ6 (13)	Контроллер флоппи-дисководов
IRQ7 (14)	Параллельный порт LPT1
IRQ8 (2)	Часы реального времени (CRT)
IRQ9 (3)	Свободен
IRQ10 (4)	Контроллер видеоадаптера
IRQ11 (5)	Свободен
IRQ12 (6)	Мышь PS/2
IRQ13 (7)	Математический сопроцессор
IRQ14 (8)	Первый контроллер жесткого диска
IRQ15 (9)	Второй контроллер жесткого диска

Чтобы получить доступ к контроллеру прерываний, следует использовать следующие порты ввода-вывода: 20h, 21h, A0h и A1h. Порты 20h и 21h обрабатывают прерывания IRQ0—IRQ7 для первого контроллера, а порты A0h и A1h — прерывания IRQ8—IRQ15 для второго контроллера прерываний.

Для организации работы контроллера прерываний используются специальные команды: управляющая команда инициализации (ICW — Initialization Command Word) и управляющая команда операции (OCW — Operation Command Word). Существуют четыре команды инициализации (от 1 до 4): ICW1, ICW2, ICW3 и ICW4. Рассмотрим каждую из них подробнее.

14.1. Команда *ICW1*

Это начальная команда инициализации контроллера прерываний. Для записи данной команды в регистры контроллеров (ведущего и ведомого) используются соответственно порты 20h и A0h. Формат команды ICW1 представлен в табл. 14.2.

Таблица 14.2. *Формат команды* ICW1

Биты	7	6	5	4	3	2	1	0
Описание	0	0	0	1	PT	Размер	Режим	ICW4

Приведем краткое описание таблицы.

- Бит 0 управляет использованием команды ICW4. Если бит установлен в 1, команда будет вызвана.
- Бит 1 определяет использование ведомого контроллера (1 — не используется, 0 — используется).
- Бит 2 определяет размер вектора прерывания (1 — 4 байта, 0 — 8 байтов).
- Бит 3 определяет режим срабатывания триггера (1 — по фронту, 0 — по уровню).
- Бит 4 должен быть установлен в 1.
- Биты 5—7 должны быть установлены в 0.

14.2. Команда *ICW2*

Эта команда позволяет установить адрес вектора прерывания для IRQ0 или IRQ8. Используются только биты с 3 по 7. Для первого контроллера нужно записать значение 08h, а для второго — 70h.

14.3. Команда *ICW3*

Команда имеет различное значение, в зависимости от типа контроллера. Для ведущего устройства используются биты 0—7. Если значение равно 1, значит, подключен ведомый контроллер. Для ведомого устройства используются только биты 0—2. Они определяют номер выхода ведущего контроллера, к которому подключен ведомый, и могут принимать одно из следующих значений: 000b — выход 0, 001b — выход 1, 010b— выход 2, 011b — выход 3, 100b — выход 4, 101b — выход 5, 110b — выход 6 и 111b — выход 7. Биты 3—7 должны быть установлены в 0.

14.4. Команда *ICW4*

Эта команда позволяет настроить дополнительные режимы работы. Формат команды представлен в табл. 14.3.

***Таблица 14.3.** Формат команды* ICW4

Биты	7	6	5	4	3	2	1	0
Описание	0	0	0	CP	Режим		Авто	CPU

Приведем краткий комментарий к таблице.

- Бит 0 определяет поддержку типа процессора (1 — 8086, 0 — 8085).
- Бит 1, установленный в 1, означает, что используется режим автоматического завершения прерывания EOI.
- Биты 2—3 определяют режим работы: 00b или 01b — для небуферизированного режима, 10b — буферизация для ведомого контроллера, 11b — буферизация для ведущего контроллера.
- Бит 4 определяет специальный вложенный режим (1 — использовать, 0 — не использовать).
- Биты 5—7 зарезервированы и должны быть установлены в 0.

Кроме команд инициализации, существуют также три команды управления: OCW1, OCW2 и OCW3. Они позволяют изменять параметры работы уже инициализированного контроллера прерываний.

14.5. Команда *OCW1*

Команда управляет маскированием различных прерываний. Установка бита в 1 позволяет запретить соответствующее прерывание, а сброс бита в 0 разрешает его. В табл. 14.4 показаны соотношения битов регистра и номеров прерываний для обоих контроллеров.

***Таблица 14.4.** Соотношения битов регистра и номеров прерываний*

Бит	7	6	5	4	3	2	1	0
Порт 21h	IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0
Порт A0h	IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8

Прежде чем изменить значение маски прерываний, необходимо прочитать текущее значение из порта, а затем, установив нужный разряд с помощью операции ИЛИ, записать полученный результат обратно в порт.

14.6. Команда *OCW2*

Данная команда позволяет управлять использованием команды EOI (окончанием прерывания), а также изменяет приоритет выполнения конечной фазы прерывания. Формат команды представлен в табл. 14.5. Команда (сигнал) EOI необходима контроллеру прерываний для завершения операции обработки прерывания и сброса внутренних регистров для последующего использования, поэтому практически всегда по окончании работы с прерыванием необходимо записывать в регистр 20h (A0h) значение 20h.

Таблица 14.5. *Формат команды* OCW2

Биты	7	6	5	4	3	2	1	0
Описание	Использование EOI			0	0	Приоритет		

Приведем краткое описание таблицы.

- Биты 0—2 определяют номер линии (IRQ), которая будет использована, если бит 6 установлен в 1. Возможны следующие значения: 000b — выход 0, 001b — выход 1, 010b — выход 2, 011b — выход 3, 100b — выход 4, 101b — выход 5, 110b — выход 6 и 111b — выход 7. Биты 3—7 должны быть установлены в 0.
- Биты 3—4 определяют код команды OCW2 (00b). Должны быть установлены в 0.
- Биты 5—7 определяют вариант использования команды окончания прерывания (EOI). Возможны следующие значения:
  - 000b — автоматический сдвиг приоритетов запрещен;
  - 100b — автоматический сдвиг приоритетов разрешен;
  - 001b — неспецифичная команда EOI;
  - 011b — специфичная команда EOI;
  - 101b — сдвиг приоритетов для неспецифичной команды EOI;
  - 111b — сдвиг приоритетов для специфичной команды EOI;
  - 110b — использование сдвига приоритетов;
  - 010b — нет никаких операций.

14.7. Команда *OCW3*

Команда позволяет прочитать текущее состояние контроллера прерываний. Формат команды представлен в табл. 14.6.

Таблица 14.6. Формат команды OCW3

Биты	7	6	5	4	3	2	1	0
Описание	0	Маска		0	1	Опрос	Статус	

Приведем краткий комментарий к таблице.

- Биты 0—1 позволяют установить режим доступа к состоянию контроллера. Возможны следующие значения: 00b и 01b — не читать состояние, 10b — читать регистр состояния на следующем прерывании, 11b — читать регистр состояния.
- Бит 2 используется для метода поллинга (опроса). Если бит установлен в 1, режим поллинга используется.
- Биты 3—4 определяют код команды OCW2 (01b).
- Биты 5—6 определяют состояние специального режима маски (00b и 01b — не использовать, 10b — выключить специальный режим маски, 11b — включить специальный режим маски).
- Бит 7 зарезервирован и должен быть установлен в 0.

Вот и вся необходимая теория. А теперь рассмотрим практические примеры программирования контроллера прерываний. Сначала напишем код для запрещения прерывания IRQ15 (второй контроллер IDE) так, как показано в листинге 14.1.

Листинг 14.1. Запрещение прерывания IRQ15

```
; сначала читаем порт A1h для получения маски прерываний
xor AL, AL   ; обнуляем регистр
in  AL, 0A0h ; получаем текущее значение
or  AL, 80h  ; запрещаем прерывание IRQ15
out 0A0h, AL ; записываем обновленное значение в порт
```

Аналогичный пример для C++ приведен в листинге 14.2.

Листинг 14.2. Запрещение прерывания IRQ15 в C++

```
// пишем функцию для запрещения прерывания от контроллера IDE
void DisableIRQ_IDE ( unsigned int uIRQ)
{
   DWORD dwResult = 0;
   if ( ( uIRQ < 14) && ( uIRQ > 15)) return;
   if ( uIRQ == 14) // первый контроллер
```

```
  {
    // получаем текущее значение
    inPort ( 0xA0, &dwResult, 1);
    dwResult |= 0x40; // запрещаем IRQ14
    // записываем обновленное значение в порт
    outPort ( 0xA0, dwResult, 1);
  }
  else // второй контроллер
  {
    // получаем текущее значение
    inPort ( 0xA0, &dwResult, 1);
    dwResult |= 0x80; // запрещаем IRQ15
    // записываем обновленное значение в порт
    outPort ( 0xA0, dwResult, 1);
  }
}
```

Между чтением и записью одноименного порта можно вставлять небольшую паузу. Попробуем запретить прерывания клавиатуры (листинг 14.3).

Листинг 14.3. Запрещение прерывания IRQ1 для клавиатуры

```
; сначала читаем порт 21h для получения маски прерываний
xor AL, AL   ; обнуляем регистр
in  AL, 21h  ; получаем текущее значение
or  AL, 02h  ; запрещаем прерывание IRQ1
; добавляем маленькую задержку
jmp short $ + 2
out 21h, AL  ; записываем обновленное значение в порт
```

Аналогичный пример для C++ приведен в листинге 14.4.

Листинг 14.4. Запрещение прерывания IRQ1 в C++

```
// пишем функцию для управления прерыванием от клавиатуры
void DisableIRQ_Kbr ( bool bEnable)
{
  DWORD dwResult = 0;
  if ( bEnable) // запретить прерывание
  {
    // получаем текущее значение
    inPort ( 0x21, &dwResult, 1);
    dwResult |= 0x02; // запрещаем IRQ1
```

```
    Sleep ( 1); // добавляем маленькую задержку
    // записываем обновленное значение в порт
    outPort ( 0x21, dwResult, 1);
  }
  else // разрешить прерывание
  {
    // получаем текущее значение
    inPort ( 0x21, &dwResult, 1);
    dwResult &= 0xFD; // разрешаем IRQ1
    // записываем обновленное значение в порт
    outPort ( 0x21, dwResult, 1);
  }
}
```

Вот и все, что может потребоваться в Windows для программирования контроллера прерываний.

ГЛАВА 15

Процессор

Думаю, нет смысла говорить о роли процессора в компьютерной системе. Эта небольшая по размерам микросхема является сердцем и мозгом компьютера и позволяет собрать в одно целое разнообразные типы устройств. В результате мы получаем мощный вычислительный комплекс, позволяющий решать не только и не столько научные задачи, но позволяющий нам окунуться в мир музыки или кино, игрушек и других всевозможных развлечений. Правда, на работе иногда приходится возиться с текстовыми процессорами и бухгалтерскими программами, рисовать однообразные логотипы и обрабатывать базы данных, но главное, что компьютер дает нам уникальную возможность самообразования и выражения своих способностей. И все это, в первую очередь, благодаря небольшой микросхеме, установленной внутри корпуса и практически бесшумно (если выкинуть медный кулер) выполняющей свой тяжкий труд.

Каждый программист должен иметь базовые сведения о центральном процессоре. К сожалению, прогресс так быстро создает новые и новые модификации, что угнаться за всем просто невозможно. В этой главе мы рассмотрим стандартные вопросы программирования CPU (Central Processing Unit — центральный процессор).

Для получения информации о процессоре в языке Assembler существует специальная команда `CPUID`. Она позволяет получить как версию процессора, так и поддерживаемые им возможности. Данную команду можно использовать для процессоров Intel (80486), AMD (80486DX4) и Cyrix M1. Прежде чем начать работу с этой командой, следует убедиться, что ее можно использовать в данной системе. Для этого необходимо установить бит `AC` в регистре `EFLAGS`. Если операция пройдет успешно, значит, команда `CPUID` поддерживается. Пример кода для проверки команды `CPUID` представлен в листинге 15.1. Если команда `CPUID` не поддерживается вашим компилятором, используйте вместо нее строку "`db 0x0F, 0xA2`".

Листинг 15.1. Проверка поддержки процессором команды CPUID

```
pushfd        ; сохраняем текущее значение регистра EFLAGS
pop EAX ,     ; получаем EFLAGS
mov ECX, EAX ; копируем
xor EAX, 40000h; устанавливаем в 1 бит AC (для ID 20000h)
push EAX      ; новое значение регистра EFLAGS
popfd         ; заменяем текущее значение EFLAGS
pushfd        ; сохраняем текущее значение регистра EFLAGS
pop EAX       ; сохраняем EFLAGS
xor EAX, ECX ; сравниваем
je  NO_CPUID ; процессор ниже 80486
```

Кроме того, для проверки можно выполнить ту же последовательность действий, но для бита ID (21 бит в EFLAGS). Аналогичный пример на C++ представлен в листинге 15.2.

Листинг 15.2. Проверка поддержки процессором команды CPUID в C++

```
// макрос для замены команды CPUID
#define CPUID _asm _emit 0x0F _asm _emit 0xA2
// пишем функцию проверки
bool IsCPUID ()
{
   __int32 i32Result = 0;
   // блока кода на ассемблере
   __asm
    {
     pushfd
     pop EAX
     mov ECX, EAX
     xor EAX, 40000h; (для ID 20000h)
     push EAX
     popfd
     pushfd
     pop EAX
     xor EAX, ECX
     je exit_proc
     mov i32Result, 1
     exit_proc:
    }
   if ( i32Result) return true; // CPUID поддерживается
```

```
    // CPUID не поддерживается
    return false;
}
```

Теперь, когда мы убедились в поддержке команды CPUID, можно получить доступную информацию об установленном процессоре. Перед выполнением команды CPUID в регистр EAX следует записать определенное числовое значение, от которого будет зависеть формат возвращаемых данных. Например, для процессоров Intel формат данных показан в табл. 15.1.

***Таблица 15.1.** Формат данных для процессора Intel*

Значение регистра EAX	Тип получаемой информации
00h	Регистр EAX: максимальное входное значение Регистр EBX: 756E6547h ("uneG") Регистр ECX: 49656E69h ("Ieni") Регистр EDX: 6C65746Eh ("letn")
01h	Регистр EAX: версия (модель, тип, семейство) Регистр EBX: зарезервирован Регистр ECX: зарезервирован Регистр EDX: информация о дополнительных свойствах
02h	Регистр EAX: информация о встроенном кэше Регистр EBX: информация о встроенном кэше Регистр ECX: информация о встроенном кэше Регистр EDX: информация о встроенном кэше

Приведем краткое описание таблицы.

- Если входное числовое значение равно 00h, в регистр EAX записывается максимальное значение, которое может использоваться (в EAX) с командой CPUID (в данном случае 2). Регистры EBX, ECX и EDX содержат составляющие имени производителя в ASCII-кодах (12 символов), причем первое значение всегда расположено в младшем регистре (BL, CL и DL). Для процессоров фирмы AMD используется строка "AuthenticAMD", а для Cyrix — "CyrixInstead".
- Если числовое значение равно 01h, в регистр EAX записывается информация о версии процессора. Формат этого регистра показан в табл. 15.2. Регистры EBX и ECX зарезервированы и не используются. В регистр EDX

записывается информация о дополнительных свойствах процессора. Формат этого регистра представлен в табл. 15.3.

- Входное числовое значение `02h` позволяет получить информацию о кэшах. В табл. 15.4 перечислены некоторые значения для определения размера внутреннего кэша процессора. При этом в регистр `EAX` записывается количество вызовов `CPUID` (`02h`), необходимое для получения всей информации.

Таблица 15.2. *Формат регистра с версией CPU*

Биты	**31—14**	**13—12**	**11—8**	**7—4**	**3—0**
Описание	Резерв	Тип	Family ID	Model ID	Stepping ID

Приведем некоторые пояснения к табл. 15.2.

- Биты 3—0 определяют модификацию процессора. Для Intel это значение будет больше 3, если установленный процессор выше 80486.
- Биты 7—4 определяют модель процессора. Значение этого поля зависит от соседнего поля Family ID и применяется совместно с ним, чтобы идентифицировать тип установленного процессора. Известные мне комбинации значений для процессоров Intel перечислены далее в табл. 15.5.
- Биты 11—8 определяют номер семейства процессора.
- Биты 13—12 для старых процессоров указывают один из следующих типов: `00b` — OEM, `01b` — Overdrive, `10b` — Dual.
- Биты 31—14 зарезервированы и не используются (должны быть равны 0).

Таблица 15.3. *Формат регистра свойств процессора*

Бит	Обозначение	Описание
0	FPU	Наличие математического сопроцессора
1	VME	Поддержка виртуального режима V86
2	DE	Поддержка точки останова для отладчика
3	PSE	Поддержка расширенных размеров страниц (до 4 Мбайт)
4	TSC	Поддержка команды RDTSC и CR4
5	MSR	Поддержка команд RDMSR и WRMSR
6	PAE	Поддержка расширенных адресов (более 32 бит)
7	MCE	Поддержка проверки машинных ошибок
8	CX8	Поддержка команды CMPXCHG8B

Таблица 15.3 (окончание)

Бит	Обозначение	Описание
9	APIC	Поддержка встроенного контроллера прерываний APIC
10	—	Резерв
11	SEP	Поддержка быстрых системных вызовов (команды SYSENTER и SYSEXIT)
12	MTRR	Поддержка регистров диапазона памяти MTRR
13	PGE	Поддержка PGE в CR4
14	MCA	Поддержка проверки машинной архитектуры
15	CMOV	Поддержка расширенных команд CMOV
16	PAT	Поддержка таблицы атрибутов, позволяющей увеличить адресуемую память
17	PSE-36	Поддержка 36-разрядных расширенных страниц (страницы размером 4 Мбайт позволяют использовать физическую адресацию памяти свыше 4 Гбайт)
18—22	—	Резерв
23	MMX	Поддержка технологии MMX
24	FXSR	Поддержка быстрых команд для чисел с плавающей точкой (FXSAVE и FXRSTOR)
25—31	—	Зарезервированы и должны быть равны 0

Таблица 15.4. *Значения второго кэша процессора*

Код	Размер кэша, Кбайт
40h	Кэш отсутствует
41h	128
42h	256
43h	512
44h	1024
45h	2048
79h	128
7Ah	512
7Bh	1024
7Ch	2048
82h	256

Таблица 15.4 (окончание)

Код	Размер кэша, Кбайт
83h	512
84h	1024
85h	2048

Таблица 15.5. *Возможные значения для полей Family ID и Model ID (Intel)*

Family ID	Model ID	Тип процессора
0000b	—	8086/8088
0001b	—	80186/80188
0010b	—	80286
0011b	0000b	Intel386 DX
	0010b	Intel386 SX (CX, EX)
	0100b	Intel386 SL
	1111b	Неизвестный тип
0100b	0000b	Intel486 DX
	0001b	Intel486 DX
	0010b	Intel486 SX
	0011b	Intel487 SX (Intel486 DX)
	0100b	Intel486 SL
	0101b	IntelSX2
	0111b	IntelDX2
	1000b	IntelDX4
	1111b	Неизвестный тип
0101b	0001b	Pentium
	0010b	Pentium P54C
	0011b	Pentium overdrive
	0101b	IntelDX4 разогнанный до Pentium
	0100b	Intel Pentium MMX
	0111b	Intel Pentium (Mobile)
	1000b	Intel Pentium MMX (Mobile)
	1111b	Неизвестный тип

Таблица 15.5 (окончание)

Family ID	Model ID	Тип процессора
0110b	0000b	Intel Pentium Pro
	0001b	Intel Pentium Pro
	0011b	Intel Pentium II
	0101b	Intel Celeron (если кэш 2 равен 0x40 или 0x41)
	0101b	Intel Pentium II (если кэш 2 равен 0x43)
	0101b	Intel Pentium II Xeon (если кэш 2 равен 0x44 или 0x45)
	0110b	Intel Celeron A (если кэш 2 равен 0x40 или 0x41)
	0110b	Intel Pentium II (если кэш 2 равен 0x42 или 0x43)
	0110b	Intel Celeron Mobile (если кэш 2 равен 0x41 и Stepping ID = 10)
	0110b	Intel Pentium II Mobile (если кэш 2 равен 0x42 или 0x82 и Stepping ID = 10)
	0111b	Intel Pentium III (если кэш 2 не равен 0x44 и 0x45)
	0111b	Intel Pentium III Xeon (если кэш 2 равен 0x44 или 0x45)
	1000b	Intel Pentium III E Coppermine (если кэш 2 не равен 0x41)
	1000b	Intel Celeron 2 (если кэш 2 равен 0x41)
	1010b	Intel Pentium III Xeon
	1111b	Неизвестный тип
1111b	0000b	Intel Pentium 4
	0001b	
	0010b	

Рассмотрим пример получения информации о процессоре (листинг 15.3).

Листинг 15.3. Получение стандартной информации о CPU

```
Vendor_Name db 12 dup (?); информация о производителе
Stepping_ID db ?; значение Stepping ID
Model_ID db ?   ; значение Model ID
Family_ID db ?  ; значение Family ID
Feature_CPU dd ?; свойства процессора
CacheCPU dd 4 dup (?)
```

```
; общий код программы
xor EAX, EAX ; устанавливаем EAX в 0
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
; сохраняем полученные данные
mov dword ptr Vendor_Name, EBX
mov dword ptr Vendor_Name[+ 4], EDX
mov dword ptr Vendor_Name[+ 8], ECX
; получаем данные о свойствах и версии
xor EAX, EAX ; устанавливаем EAX в 0
inc EAX      ; определяем значение 1
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
mov Stepping_ID, AL; копируем первый байт регистра EAX
and Stepping_ID, 0Fh; выделяем значение Stepping ID
and AL, 0F0h ; восстанавливаем значение регистра
shr AL, 4    ; сдвигаем на 4 разряда вправо
mov Model_ID, AL; получаем значение Model ID
and EAX, 0FF0h; выделяем биты 8-11
shr EAX, 8   ; сдвигаем на 8 разрядов вправо
and EAX, 0Fh ; выделяем значение Family_ID
; получаем дополнительные свойства процессора
mov Feature_CPU, ECX
; получаем размер встроенного кэша
xor EAX, EAX ; устанавливаем EAX в 0
mov EAX, 02h ; определяем значение 02h
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
mov dword ptr [CacheCPU + 0], EAX
mov dword ptr [CacheCPU + 4], EBX
mov dword ptr [CacheCPU + 8], ECX
mov dword ptr [CacheCPU + 12], EDX
```

Аналогичный пример на C++ представлен в листинге 15.4.

Листинг 15.4. Получение стандартной информации о CPU в C++

```
// макрос для замены команды CPUID
#define CPUID _asm _emit 0x0F _asm _emit 0xA2
// пишем функцию для получения стандартных сведений о процессоре
void GetCPU_Info ()
{
   BYTE Vendor_Name[13];
   BYTE Stepping_ID;
   BYTE Model_ID;
   BYTE Family_ID;
```

```
DWORD Feature_CPU;
DWORD CacheCPU[4];
unsigned int uCache_1 = 0, uCache_2 = 0;
// получаем информацию о процессоре
__asm
{
  xor EAX, EAX; устанавливаем EAX в 0
  cpuid; вызываем команду CPUID
 ; сохраняем полученные данные
  mov dword ptr Vendor_Name, EBX
  mov dword ptr Vendor_Name[+ 4], EDX
  mov dword ptr Vendor_Name[+ 8], ECX
 ; получаем данные о свойствах и версии
  xor EAX, EAX; устанавливаем EAX в 0
  inc EAX; определяем значение 1
  cpuid; вызываем команду CPUID
  mov Stepping_ID, AL; копируем первый байт регистра EAX
  and Stepping_ID, 0Fh; выделяем значение Stepping ID
  and AL, 0F0h; восстанавливаем значение регистра
  shr AL, 4 ; сдвигаем на 4 разряда вправо
  mov Model_ID, AL; получаем значение Model ID
  and EAX, 0FF0h; выделяем биты 8-11
  shr EAX, 8; сдвигаем на 8 разрядов вправо
  and EAX, 0Fh; выделяем значение Family_ID
 ; получаем дополнительные свойства процессора
  mov Feature_CPU, ECX
 ; получаем размер встроенного кэша
  xor EAX, EAX; устанавливаем EAX в 0
  mov EAX, 02h; определяем значение 02h
  cpuid; вызываем команду CPUID
  mov dword ptr [CacheCPU + 0], EAX
  mov dword ptr [CacheCPU + 4], EBX
  mov dword ptr [CacheCPU + 8], ECX
  mov dword ptr [CacheCPU + 12], EDX
}
 // обрабатываем значение кэша для Intel
 DWORD cacheTemp = CacheCPU[3];
 BYTE cache_L1 = ( BYTE) ( cacheTemp >> 8); // размер первого кэша
 BYTE cache_Tmp = ( BYTE) ( cacheTemp >> 24);
 BYTE cache_L2 = ( BYTE) ( cacheTemp >> 0); // размер второго кэша
 // получаем размер первого кэша
 if ( ( cache_Tmp == 0x0A) && ( cache_L1 == 0x06))
   uCache_1 = 16; // 16 Кб
```

```
    else if ( ( cache_Tmp == 0x0C) && ( cache_L1 == 0x08))
      uCache_1 = 32; // 32 Кб
    // получаем размер второго кэша
    switch ( cache_L2)
    {
     case 0x40: // нет второго кэша
      break;
     case 0x41: // 128 Кб
     case 0x79:
      uCache_2 = 128;
      break;
     case 0x42: // 256 Кб
     case 0x82:
      uCache_2 = 256;
      break;
     case 0x43: // 512 Кб
     case 0x7A:
     case 0x83:
      uCache_2 = 512;
      break;
     case 0x44: // 1024 Кб
     case 0x7B:
     case 0x84:
      uCache_2 = 1024;
      break;
     case 0x45: // 2048 Кб
     case 0x7C:
     case 0x85:
      uCache_2 = 2048;
      break;
    }
}
```

Кроме стандартных аргументов, процессор может поддерживать расширенные значения для команды CPUID, перечисленные в табл. 15.6. Эти значения следует использовать для получения данных о совместимых с Intel процессоров AMD и Cyrix.

Таблица 15.6. *Расширенные значения для команды CPUID*

Код	Описание
03h	Позволяет получить серийный номер процессора (начиная с Intel Pentium III), закодированный в регистрах ECX и EDX

Таблица 15.6 (окончание)

Код	Описание
80000000h	Максимальное расширенное значение, поддерживаемое процессором
80000001h	Возвращает версию и свойства для процессора
80000002h	Название процессора
80000003h	Название процессора
80000004h	Название процессора
80000005h	Возвращает информацию о размерах первого кэша
80000006h	Возвращает информацию о размерах второго кэша

В листинге 15.5 приводится пример использования расширенного значения команды CPUID для получения серийного номера процессора.

Листинг 15.5. Получение серийного номера процессора

```
; выделяем переменную для хранения серийного номера
Intel_SerialNumber dd 3 dup (0)
; общий код программы
xor EAX, EAX ; обнуляем регистр
cpuid        ; вызываем команду CPUID
mov dword ptr [Intel_SerialNumber + 0], EAX
mov EAX, 03h ; получить номер CPU
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
; получаем серийный номер
mov dword ptr [Intel_SerialNumber + 4], EDX
mov dword ptr [Intel_SerialNumber + 8], ECX
```

Аналогичный пример на C++ представлен в листинге 15.6.

Листинг 15.6. Получение серийного номера процессора в C++

```
// макрос для замены команды CPUID
#define CPUID _asm _emit 0x0F _asm _emit 0xA2
// выделяем переменную для хранения серийного номера
DWORD dwSerialNumber[3];
// пишем функцию
void GetSN_Intel ()
{
   __asm
```

```
   {
     xor EAX, EAX; обнуляем регистр
     cpuid; вызываем команду CPUID
     mov dword ptr [dwSerialNumber + 0], EAX
     mov EAX, 03h; получить номер CPU
     cpuid; вызываем команду CPUID
    ; получаем серийный номер
     mov dword ptr [dwSerialNumber + 4], EDX
     mov dword ptr [dwSerialNumber + 8], ECX
   }
}
// сохраняем серийный номер в строку
char szSerial[30];
sprintf ( szSerial, "Intel Serial Number: %08lx-%08lx-%08lx",
          dwSerialNumber[0], dwSerialNumber[1], dwSerialNumber[2]);
```

И еще попробуем получить название процессора, используя расширенные значения команды `CPUID` (листинг 15.7).

Листинг 15.7. Получение названия установленного процессора

```
; выделяем переменную для хранения имени процессора
CPU_Name db 48 dup (0)
; общий код программы
xor EAX, EAX ; обнуляем регистр
mov EAX, 80000002h
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
mov dword ptr [CPU_Name + 0], EAX
mov dword ptr [CPU_Name + 4], EBX
mov dword ptr [CPU_Name + 8], ECX
mov dword ptr [CPU_Name + 12], EDX
mov EAX, 80000003h
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
mov dword ptr [CPU_Name + 16], EAX
mov dword ptr [CPU_Name + 20], EBX
mov dword ptr [CPU_Name + 24], ECX
mov dword ptr [CPU_Name + 28], EDX
mov EAX, 80000004h
cpuid        ; вызываем команду CPUID или заменяем на db 0x0F, 0xA2
mov dword ptr [CPU_Name + 32], EAX
mov dword ptr [CPU_Name + 36], EBX
mov dword ptr [CPU_Name + 40], ECX
mov dword ptr [CPU_Name + 44], EDX
```

Аналогичный пример на C++ представлен в листинге 15.8.

Листинг 15.8. Получение названия установленного процессора в C++

```
// макрос для замены команды CPUID
#define CPUID _asm _emit 0x0F _asm _emit 0xA2
// выделяем переменную для хранения серийного номера
char szCPU_Name[49];
szCPU_Name[48] = 0; // завершающий ноль
// пишем функцию
void GetCPU_Name ()
{
   __asm
   {
     xor EAX, EAX; обнуляем регистр
     mov EAX, 80000002h
     cpuid; вызываем команду CPUID
     mov dword ptr [CPU_Name + 0], EAX
     mov dword ptr [CPU_Name + 4], EBX
     mov dword ptr [CPU_Name + 8], ECX
     mov dword ptr [CPU_Name + 12], EDX
     mov EAX, 80000003h
     cpuid; вызываем команду CPUID
     mov dword ptr [CPU_Name + 16], EAX
     mov dword ptr [CPU_Name + 20], EBX
     mov dword ptr [CPU_Name + 24], ECX
     mov dword ptr [CPU_Name + 28], EDX
     mov EAX, 80000004h
     cpuid; вызываем команду CPUID
     mov dword ptr [CPU_Name + 32], EAX
     mov dword ptr [CPU_Name + 36], EBX
     mov dword ptr [CPU_Name + 40], ECX
     mov dword ptr [CPU_Name + 44], EDX
   }
}
```

Вот и все, что мне хотелось рассказать о программировании процессора.

ГЛАВА 16

Аппаратный мониторинг системы

В конце 90-х годов производители материнских плат стали активно использовать различные датчики температуры, питания и управления вентиляторами, благодаря чему появилась возможность контролировать критические компоненты системы в реальном времени. В настройки BIOS были добавлены дополнительные возможности мониторинга напряжений блока питания, температуры центрального процессора и внутреннего пространства корпуса, а также вывода информации о текущем состоянии вентиляторов (до трех). Важность этого трудно переоценить. Кроме контроля за состоянием системы, пользователь может самостоятельно устанавливать ограничения на температуру или питающие напряжения, что при умелом использовании помогает в ранней диагностике аппаратных сбоев оборудования. Например, установив максимальное значение температуры разогрева для процессора, можно защитить его от перегревания и возможного выхода из строя (это особенно актуально для процессоров фирмы AMD). Контроль за вентилятором процессора позволяет вовремя выключить компьютер, если вентилятор вдруг заклинил или вовсе сгорел. А продвинутые пользователи, например, знают, что импульсный блок питания имеет довольно короткий срок эксплуатации (примерно 5 лет) и требует особого внимания. Необходимо постоянно (хотя бы раз в месяц) проверять значения питающих напряжений и при отклонении последних более чем на 10 % в ту или иную сторону сразу же заменять блок питания целиком. Как правило, ремонту импульсные блоки питания не подлежат (не верьте никому, кто уверяет вас в обратном), и несоблюдение этих правил может привести к выходу из строя всех (!) компонентов системы. Но даже потеря "железа" не так важна, как потеря информации, хранящейся на компьютере.

Из всего сказанного следует, что программист должен знать и уметь работать с перечисленными выше возможностями для повышения не только безопасности компьютерной системы, но и своего профессионального мастерства.

В этой главе мы поговорим о том, как в современных компьютерах реализуется аппаратный мониторинг оборудования.

* * *

Для поддержки мониторинга системы на материнскую плату добавляют самостоятельный модуль (микросхему) управления, позволяющий эффективно отслеживать напряжения питания, обороты вентиляторов и температуру. Существуют различные модули контроля оборудования, выпускаемые ведущими производителями, но здесь мы рассмотрим наиболее популярные микросхемы фирмы National Semiconductor. Изучив использование чипов данной фирмы, вы без особого труда сможете разобраться с остальными.

Итак, имеется несколько основных микросхем аппаратного мониторинга системы: LM78, LM79 и LM80. Они практически идентичны, различаются в основном диапазоном и точностью измеряемых параметров. Мы рассмотрим только вторую и третью микросхемы, поскольку LM78 практически не отличается от LM79. Основные характеристики этих микросхем представлены в табл. 16.1.

Таблица 16.1. *Основные характеристики микросхем LM79 и LM80*

Описание параметра	Поддерживаемое значение	
	LM79	LM80
Напряжение питания	5 В	2,8—5,75 В
Потребляемый ток (работа–простой)	1—10 мкА	0,2—15 мкА
Разрядность АЦП	8 бит	8 бит
Максимальная погрешность измерения напряжения	±1 %	±1 %
Погрешность измерения температуры	Для диапазона от –10 до +100 °С равна ±3 °С	Для диапазона от –25 до +125 °С равна ±3 °С
Точность измерения температуры	—	0,5 °С

Микросхема LM79 работает с шинами ISA и Serial Bus и поддерживает следующие возможности:

- контроль температуры;
- управление тремя вентиляторами;
- пять положительных входных напряжений;
- два отрицательных входных напряжения;
- проверка на сравнение получаемых значений;

- дополнительный температурный датчик (например, LM75 или LM99);
- контроль вскрытия системного блока или изъятия какого-либо модуля.

Данная микросхема считывает обороты для первого и второго вентилятора в диапазоне от 1 100 до 8 800 оборотов в минуту, а для третьего фиксирует только среднее значение 4 400.

Микросхема LM80 работает только с шиной Serial Bus и поддерживает такие же возможности, как и предыдущая.

Чтобы получить доступ к микросхеме мониторинга можно использовать порт `290h`. Однако на несовместимой с Intel платформе номер порта может быть другим (за информацией следует обратиться на сайт производителя материнской платы).

Есть еще один нюанс, о котором следует знать. Официально в документации Intel указан порт номер `80h`, хотя реально работает `290h`. Для определения порта шины Serial Bus (если стоит микросхема, отличная от LM78 и LM79) необходимо применить процедуру поиска на шине PCI *(см. гл. 10)* или попытаться обратиться через порты `90h` (базовый регистр адреса) и `D2h` (управляющий регистр).

Микросхемы мониторинга используют для управления и настройки определенные внутренние регистры, представленные в табл. 16.2. В скобках указаны адреса регистров для микросхемы LM80.

***Таблица 16.2.** Внутренние регистры управления*

Адрес регистра	Значение по умолчанию	Название	Описание
40h (00h)	00001000b	Регистр конфигурации	Позволяет настроить параметры работы
41h (01h)	00000000b	Регистр состояния 1	Отслеживают превышения допустимых заданных пределов или события прерываний (после чтения первого регистра будет автоматически выбран второй)
42h (02h)	00000000b	Регистр состояния 2	
43h (03h)	00000000b	Регистр маски 1 (SMI)	Позволяют маскировать различные источники прерываний (после чтения первого регистра будет автоматически выбран второй). Регистры SMI (System Management Interrupt) управляют системными прерываниями, а регистры NMI (Non-Maskable Interrupt) — немаскируемыми прерываниями
44h (04h)	00000000b	Регистр маски 2 (SMI)	
45h (—)	00000000b	Регистр маски 1 (NMI)	
46h (—)	01000000b	Регистр маски 2 (NMI)	

Таблица 16.2 (окончание)

Адрес регистра	Значение по умолчанию	Название	Описание
47h (05h)	0101xxxxb (00010100b)	Регистр делителя	Позволяет читать значение делителя для установки оборотов первого и второго вентиляторов (младшие 4 бита хранят состояние делителя, а старшие 4 бита позволяют установить новое значение). Для LM80 хранит одно значение делителя, равное 2 (4 400)
48h (—)	00101101b	Регистр адреса шины Serial Bus	Содержит адрес шины Serial Bus (по умолчанию 2Dh) и может быть изменен
49h (—)	1100000xb	Регистр сброса	Позволяет выполнить сброс микросхемы, установив для регистров значения по умолчанию
— (06h)	—	Регистр температуры	Управляет представлением значений температуры (если бит 3 равен 1, то используется 12-разрядное представление температуры и 4 младших бита значения расположены в битах 4—7)
20h—3Fh	—	Регистры данных	Позволяют получить или установить все поддерживаемые параметры

После подачи питания на микросхемы начинает работать непрерывный цикл измерения параметров системы с частотой один раз в секунду. Если микросхема содержит настройки для предельно-допустимых ограничений, происходит сравнение полученных и заданных значений. Если полученное значение превышает заданное, микросхема устанавливает сигнал прерывания в соответствующий регистр состояния. С помощью регистров маски устанавливается генерация тех или иных прерываний во время мониторинга оборудования, а регистр конфигурации позволяет заблокировать те или иные прерывания.

Поскольку микросхема LM79 поддерживает шину ISA, имеются также четыре внешних регистра, необходимые для управления всеми внутренними регистрами LM79 (табл. 16.3).

Работа с LM79 для ISA организуется следующим образом:

1. В регистр x5h записывается номер внутреннего регистра (см. табл. 16.2).
2. После этого можно читать или писать данные в регистр x6h.

Таблица 16.3. *Внешние регистры управления для LM79*

Адрес	Описание
x0h	Самотестирование микросхемы
x4h	Самотестирование микросхемы
x5h	Регистр адреса для выбора внутреннего регистра (чтение и запись)
x6h	Регистр данных (чтение и запись)

Если на компьютере присутствуют обе шины (ISA и Serial Bus), следует в начале работы прочитать регистр x5h. Если бит 7 равен 0, можно работать с шиной ISA, иначе используется шина Serial Bus. Переход на шину ISA требует примерно 10 мкс времени, а обратно — всего 1 мкс. Бит 7 можно только читать.

Работа с LM79 для Serial Bus организуется следующим образом:

1. В регистр x5h записывается адрес шины Serial Bus (48h).
2. В этот же регистр пишется номер внутреннего регистра микросхемы (см. табл. 16.2).
3. В этот же регистр пишется байт данных.

Чтение данных происходит так:

1. В регистр x5h записывается адрес шины Serial Bus (48h). Этот пункт можно пропустить, если порт уже был выбран предыдущей операцией.
2. Из этого же регистра читается номер внутреннего регистра микросхемы (см. табл. 16.2).
3. Из этого же регистра читается байт данных.

Как уже говорилось ранее, адрес регистра для шины Serial Bus можно изменить, записав новое значение во внутренний регистр 48h.

Регистр конфигурации (40h) управляет сбросом устройства и блокированием прерываний. Установка бита 0 в 0 позволяет выполнить сброс микросхемы и перевести ее в энергосберегающий режим. Установка бита 1 в 1 выполняет инициализацию регистра. Бит 2 управляет блокировкой немаскируемых прерываний (1 — разрешить NMI). Бит 3 управляет работой микросхемы (0 — включить мониторинг оборудования). При начале опроса параметров бит 3 устанавливается в 0, а бит 0 — в 1. Опрос параметров ведется в следующем порядке: температура, положительные напряжения питания (линии от 0 до 4), отрицательные напряжения питания (линии 5 и 6), первый, второй и третий вентиляторы. Из-за задержки в работе микросхемы при чтении каждого последующего параметра следует делать паузу не менее 120 мс. Если за один раз вы читаете все параметры, перед следующей операцией чтения необходимо выполнить паузу не менее 1,5 сек.

Для управления вентиляторами используется внутренний генератор (22,5 кГц для выдачи сигнала за время одного полного оборота) и 8-разрядный счетчик (максимум 255 отсчетов), определяющий количество сигналов. Например, если значение счетчика равно 153, обороты вентилятора равны 4 400. Управление оборотами сводится к установке значения делителя (1, 2, 4 и 8) в регистре делителя (47h). Это будет работать только для первого и второго вентиляторов. Третий вентилятор имеет постоянные значения: делитель равен 2, а обороты — 4 400. По умолчанию начальные значения делителя и числа оборотов равны 2 и 4 400 соответственно. Для подсчета текущего значения счетчика используется следующая формула:

значение счетчика = (1,35 * 10^6) / (число оборотов * делитель).

Значения температуры кодируются 8-битным значением (LM79), где младший бит определяет 1 °С. В табл. 16.4 представлены стандартные значения регистра температуры.

Таблица 16.4. *Стандартные значения для 8-разрядного представления температуры*

Значение регистра	Температура, °С
01111101b (7Dh)	+125
00011001b (19h)	+25
00000001b (01h)	+1,0
00000000b (00h)	+0
11111111b (FFh)	–1
11100111b (E7h)	–25
11001001b (C9h)	–55

Если температура выше или ниже заданной, генерируется прерывание, и оно остается активным, пока не будет прочитан регистр состояния 1 (41h).

В микросхеме LM80 значения температуры могут кодироваться дополнительно 9- (табл. 16.5) и 12-разрядным (табл. 16.6) двоичным значением, что позволяет повысить точность измерений. Для 9-разрядного представления температуры значение младшего бита равно 0,5 °С, а для 12-разрядного — 0,0625 °С.

Младшие 8 бит значения (для 12-битного представления) температуры можно прочитать из внутреннего регистра 28h. Остаток будет записан в регистр 06h (биты 4—7). При 9-битном значении бит 9 будет записан в бит 7 регистра температуры.

Таблица 16.5. Стандартные значения для 9-разрядного представления температуры

Значение регистра	Температура, °С
011111010b (FAh)	+125
000110010b (32h)	+25
000000011b (03h)	+1,5
00000000b (00h)	+0
111111111b (1FFh)	–0,5
111001110b (1CEh)	–25
110010010b (192h)	–55

Таблица 16.6. Стандартные значения для 12-разрядного представления температуры

Значение регистра	Температура, °С
011111000000b (7D0h)	+125
000110010000b (190h)	+25
000000010000b (010h)	+1,0
000000000001b (01h)	+0,0625
00000000b (00h)	+0
111111111111b (FFFh)	–0,0625
111111110000 (FF0h)	–1,0
111001110000 (E70h)	–25
110010010000b (C90h)	–55

И последнее, что мы рассмотрим здесь, — это структура внешних и некоторых внутренних регистров управления. В табл. 16.7 и 16.8 показаны форматы регистров x5h и x6h для LM79.

В табл. 16.9 показан формат регистр выбора адреса для LM80.

В табл. 16.10 показан формат регистров конфигурации для LM79 и LM80. Они позволяют инициализировать устройство мониторинга и установить режим работы. Кроме того, прочитав значение регистра после инициализации (оно должно быть равно 08h), можно сделать вывод о наличии микросхемы мониторинга на данной системе.

Таблица 16.7. *Формат адресного регистра x5h для LM79*

Биты	Доступ	Описание
0—6	Запись и чтение	Содержит адрес внутреннего регистра (см. табл. 16.2)
7	Только чтение	Определяет готовность устройства (если бит равен 1, можно записывать данные в регистр x6h для шины Serial Bus, иначе следует писать или читать регистр x6h после завершения работы на шине Serial Bus)

Таблица 16.8. *Формат адресного регистра x6h для LM79*

Биты	Доступ	Описание
0—7	Запись и чтение	Данные, которые надо передать или получить

Таблица 16.9. *Формат адресного регистра для LM80*

Биты	Доступ	Описание
0—7	Запись и чтение	Содержит адрес внутреннего регистра (см. табл. 16.2) от 00h до 06h и от 20h до 3Fh

Таблица 16.10. *Формат регистра конфигурации для LM79 и LM80*

Биты	Доступ	Описание
0	Запись и чтение	Управление запуском мониторинга системы (1 — включить контроль, 0 — перевести в дежурный режим). Перед установкой бита в 1 следует записать желаемые параметры в регистры данных 20h—3Fh
1	Запись и чтение	Установка бита в 1 позволяет включить прерывания (для LM79 SMI-прерывания)
2	Запись и чтение	Выбор активной выходной линии для LM80 и включение прерывание NMI для LM79
3	Запись и чтение	Блокировка прерывания SMI и NMI для LM79 и IRQ для LM80 (1 — блокировать, 0 — разрешить)
4	Запись и чтение	Сброс устройства
5	Запись и чтение	Выбор NMI-прерываний (1 — NMI, 0 — IRQ)
6	Запись и чтение	Управление питанием
7	Запись и чтение	Восстановление питания на шине и инициализирует устройство

В табл. 16.11 и 16.12 показан формат первых регистров состояния (41h или 01h) для LM79 и LM80. Регистры состояния позволяют отслеживать изменение значений на заданных линиях.

Таблица 16.11. Формат первого регистра состояния для LM79

Биты	Доступ	Описание
0	Только чтение	Линия 0. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
1	Только чтение	Линия 1. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
2	Только чтение	Линия 2. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
3	Только чтение	Линия 3. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
4	Только чтение	Температура. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
5	Только чтение	Установка бита в 1 указывает то, что произошло прерывание от датчика температуры (LM75)
6	Только чтение	Первый вентилятор. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
7	Только чтение	Второй вентилятор. Установка бита в 1 указывает на выход текущего значения из заданного диапазона

Таблица 16.12. Формат первого регистра состояния для LM80

Биты	Доступ	Описание
0	Только чтение	Линия 0. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
1	Только чтение	Линия 1. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
2	Только чтение	Линия 2. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
3	Только чтение	Линия 3. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
4	Только чтение	Линия 4. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
5	Только чтение	Линия 5. Установка бита в 1 указывает на выход текущего значения из заданного диапазона

Таблица 16.12 (окончание)

Биты	Доступ	Описание
6	Только чтение	Линия 6. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
7	Только чтение	Установка бита в 1 указывает, что на вход поступил сигнал прерывания

В табл. 16.13 и 16.14 показан формат вторых регистров состояния (42h или 02h) для LM79 и LM80. Они выполняют те же функции, что и первые регистры состояния.

***Таблица 16.13.** Формат второго регистра состояния для LM79*

Биты	Доступ	Описание
0	Только чтение	Линия 4. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
1	Только чтение	Линия 5. Установка бита в 1 указывает на выход текущего значения из заданного диапазона (отрицательные напряжения)
2	Только чтение	Линия 6. Установка бита в 1 указывает на выход текущего значения из заданного диапазона (отрицательные напряжения)
3	Только чтение	Третий вентилятор. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
4	Только чтение	Установка бита в 1 указывает на вскрытие корпуса или изъятие из разъема заданного модуля
5	Только чтение	Переполнение FIFO (через порты x0h и x4h)
6	Только чтение	Дополнительный цифровой вход. Установка бита в 1 указывает, что сигнал прерывания NMI на входе отсутствует
7	Только чтение	Резерв

***Таблица 16.14.** Формат второго регистра состояния для LM80*

Биты	Доступ	Описание
0	Только чтение	Температура. Установка бита в 1 указывает на выход текущего значения из заданного диапазона (режим установлен во втором регистре маски битом 6)

Таблица 16.14 (окончание)

Биты	Доступ	Описание
1	Только чтение	Температура. Установка бита в 1 указывает, что произошло прерывание на входе для дополнительного датчика температуры (например, LM75 или LM99)
2	Только чтение	Первый вентилятор. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
3	Только чтение	Второй вентилятор. Установка бита в 1 указывает на выход текущего значения из заданного диапазона
4	Только чтение	Установка бита в 1 указывает на вскрытие корпуса или изъятие из разъема заданного модуля
5	Только чтение	Температура. Установка бита в 1 указывает на выход текущего значения из заданного диапазона (режим установлен во втором регистре маски битом 7)
6	Только чтение	Резерв
7	Только чтение	Резерв

В табл. 16.15 и 16.16 показан формат первых регистров маски SMI (43h или 03h) для LM79 и LM80. Они позволяют отключить генерацию прерываний для заданной линии. Эти регистры удобно использовать для временного выключения мониторинга за отдельными линиями (например, температуры или напряжения питания), а также для получения текущего состояния блокировок.

Таблица 16.15. *Формат первого регистра маски для LM79*

Биты	Доступ	Описание
0	Запись и чтение	Линия 0. Установка бита в 1 блокирует прерывание для данной линии
1	Запись и чтение	Линия 1. Установка бита в 1 блокирует прерывание для данной линии
2	Запись и чтение	Линия 2. Установка бита в 1 блокирует прерывание для данной линии
3	Запись и чтение	Линия 3. Установка бита в 1 блокирует прерывание для данной линии
4	Запись и чтение	Температура. Установка бита в 1 блокирует прерывание для данной линии
5	Запись и чтение	Первый вентилятор. Установка бита в 1 блокирует прерывание для данной линии

Таблица 16.15 (окончание)

Биты	Доступ	Описание
6	Запись и чтение	Второй вентилятор. Установка бита в 1 блокирует прерывание для данной линии
7	Запись и чтение	Резерв

Таблица 16.16. *Формат первого регистра маски для LM80*

Биты	Доступ	Описание
0	Запись и чтение	Линия 0. Установка бита в 1 блокирует прерывание для данной линии
1	Запись и чтение	Линия 1. Установка бита в 1 блокирует прерывание для данной линии
2	Запись и чтение	Линия 2. Установка бита в 1 блокирует прерывание для данной линии
3	Запись и чтение	Линия 3. Установка бита в 1 блокирует прерывание для данной линии
4	Запись и чтение	Линия 4. Установка бита в 1 блокирует прерывание для данной линии
5	Запись и чтение	Линия 5. Установка бита в 1 блокирует прерывание для данной линии
6	Запись и чтение	Линия 6. Установка бита в 1 блокирует прерывание для данной линии
7	Запись и чтение	Установка бита в 1 блокирует прерывание на цифровом входе для любого подключенного устройства

В табл. 16.17 и 16.18 показан формат вторых регистров маски SMI (`44h` или `04h`) для LM79 и LM80. Регистры выполняют те же функции, что и первые два регистра маски.

Таблица 16.17. *Формат второго регистра маски для LM79*

Биты	Доступ	Описание
0	Запись и чтение	Линия 4. Установка бита в 1 блокирует прерывание для данной линии
1	Запись и чтение	Линия 5. Установка бита в 1 блокирует прерывание для данной линии (отрицательные напряжения)

Таблица 16.17 (окончание)

Биты	Доступ	Описание
2	Запись и чтение	Линия 6. Установка бита в 1 блокирует прерывание для данной линии (отрицательные напряжения)
3	Запись и чтение	Третий вентилятор. Установка бита в 1 блокирует прерывание для данной линии
4	Запись и чтение	Вскрытие корпуса или изъятие из разъема заданного модуля. Установка бита в 1 блокирует прерывание для данной линии
5	Запись и чтение	Переполнение FIFO. Установка бита в 1 блокирует прерывание для данной линии
6	Запись и чтение	Дополнительный цифровой вход. Установка бита в 1 блокирует прерывание для данной линии
7	Запись и чтение	Сброс. Установка этого бита в 1 позволяет сбросить настройки для регистра конфигурации

Таблица 16.18. *Формат второго регистра маски для LM80*

Биты	Доступ	Описание
0	Запись и чтение	Температура. Установка бита в 1 блокирует прерывание для данной линии
1	Запись и чтение	Температура. Установка бита в 1 блокирует прерывание на входной линии для дополнительного датчика температуры (например, LM75 или LM99)
2	Запись и чтение	Первый вентилятор. Установка бита в 1 блокирует прерывание для данной линии
3	Запись и чтение	Второй вентилятор. Установка бита в 1 блокирует прерывание для данной линии
4	Запись и чтение	Вскрытие корпуса или изъятие из разъема заданного модуля. Установка бита в 1 блокирует прерывание для данной линии
5	Запись и чтение	Температура (12-разрядное представление). Установка бита в 1 блокирует прерывание для данной линии
6	Запись и чтение	Режим прерывания по температуре. При установке в ноль прерывание генерируется, если нарушен заданный диапазон, а сигнал прерывания будет сброшен. Если значение температуры остается вне диапазона, прерывание будет выдано повторно. Если бит установлен в 1, то прерывание будет выдано только один раз при выходе значения температуры из заданного диапазона

Таблица 16.18 (окончание)

Биты	Доступ	Описание
7	Запись и чтение	То же самое, но для 12-разрядного представления температуры

В табл. 16.19 и 16.20 показан формат регистров маски NMI (`45h` и `46h`) для LM79. Регистр выполняет те же функции, что и первый регистр маски SMI.

Таблица 16.19. *Формат первого регистра маски NMI для LM79*

Биты	Доступ	Описание
0	Запись и чтение	Линия 0. Установка бита в 1 блокирует прерывание для данной линии
1	Запись и чтение	Линия 1. Установка бита в 1 блокирует прерывание для данной линии
2	Запись и чтение	Линия 2. Установка бита в 1 блокирует прерывание для данной линии
3	Запись и чтение	Линия 3. Установка бита в 1 блокирует прерывание для данной линии
4	Запись и чтение	Температура. Установка бита в 1 блокирует прерывание для данной линии
5	Запись и чтение	Температура. Установка бита в 1 блокирует прерывание на входной линии для дополнительного датчика температуры (например, LM75 или LM99)
6	Запись и чтение	Первый вентилятор. Установка бита в 1 блокирует прерывание для данной линии
7	Запись и чтение	Второй вентилятор. Установка бита в 1 блокирует прерывание для данной линии

Таблица 16.20. *Формат второго регистра маски NMI для LM79*

Биты	Доступ	Описание
0	Запись и чтение	Линия 4. Установка бита в 1 блокирует прерывание для данной линии
1	Запись и чтение	Линия 5. Установка бита в 1 блокирует прерывание для данной линии (отрицательные напряжения)
2	Запись и чтение	Линия 6. Установка бита в 1 блокирует прерывание для данной линии (отрицательные напряжения)

Таблица 16.20 (окончание)

Биты	Доступ	Описание
3	Запись и чтение	Третий вентилятор. Установка бита в 1 блокирует прерывание для данной линии
4	Запись и чтение	Вскрытие корпуса или изъятие из разъема заданного модуля. Установка бита в 1 блокирует прерывание для данной линии
5	Запись и чтение	Переполнение FIFO. Установка бита в 1 блокирует прерывание для данной линии
6	Запись и чтение	Дополнительный цифровой вход. Установка бита в 1 блокирует прерывание для данной линии
7	Запись и чтение	Сброс. Установка этого бита в 1 на время более 20 мс позволяет сбросить блокировку для вскрытия корпуса

В табл. 16.21 показан формат регистра делителя (47h) для LM79. Регистр позволяет установить значения делителя для управления оборотами вентиляторов.

Таблица 16.21. *Формат регистра делителя для LM79*

Биты	Доступ	Описание
0—3	Только чтение	Для некоторых процессоров Pentium сюда записывается напряжение питания (младшие 4 бита)
4—5	Запись и чтение	Первый вентилятор. Возможны следующие значения: 00b — 1, 01b — 2, 10b — 4 и 11b — 8
6—7	Запись и чтение	Второй вентилятор. Возможны следующие значения: 00b — 1, 01b — 2, 10b — 4 и 11b — 8

В табл. 16.22 показан формат регистра делителя (05h) для LM80. Регистр предназначен для настройки оборотов вентиляторов и установки 12-разрядного представления для температуры.

Таблица 16.22. *Формат регистра делителя для LM80*

Биты	Доступ	Описание
0	Запись и чтение	Выбор первого вентилятора. Установка бита в 1 позволяет выбрать уровень чувствительности вентилятора, а запись 0 — прочитать значение делителя

Таблица 16.22 (окончание)

Биты	Доступ	Описание
1	Запись и чтение	Выбор второго вентилятора. Установка бита в 1 позволяет выбрать уровень чувствительности вентилятора, а запись 0 — прочитать значение делителя
2—3	Запись и чтение	Установка делителя для первого вентилятора. Возможны следующие значения: 00b — 1, 01b — 2, 10b — 4 и 11b — 8
4—5	Запись и чтение	Установка делителя для второго вентилятора. Возможны следующие значения: 00b — 1, 01b — 2, 10b — 4 и 11b — 8
6	Запись и чтение	Выбор режима для 12-разрядного представления температуры (бит должен быть установлен в 1)
7	Запись и чтение	Резерв

В табл. 16.23 показан формат регистра Serial Bus (48h) для LM79. Регистр позволяет указать адрес регистра на шине Serial Bus.

***Таблица 16.23.** Формат регистра Serial Bus для LM79*

Биты	Доступ	Описание
0—6	Запись и чтение	Адрес регистра Serial Bus
7	Только чтение	Резерв

В табл. 16.24 показан формат регистра сброса (49h) для LM79. Регистр позволяет выполнить сброс микросхемы. После включения питания в регистре записано значение 1100000xb.

***Таблица 16.24.** Формат регистра Serial Bus для LM79*

Биты	Доступ	Описание
0	Только чтение	Старший бит (5) для напряжения питания процессора
1—4	Только чтение	Резерв
5	Запись и чтение	Установка бита в 1 позволит выполнить сброс микросхемы
6	Только чтение	Резерв
7	Только чтение	Идентификатор LM79 (как и для LM78 равен 0)

В табл. 16.25 показан формат регистра температуры (06h) для LM80. Регистр управляет чувствительностью измерения температуры. После включения питания в регистре записано значение 00000001b.

Таблица 16.25. Формат регистра температуры для LM80

Биты	Доступ	Описание
0	Только чтение	Состояние режима 12-разрядного представления температуры
1	Запись и чтение	Полярность температуры (1 — выше нуля, 0 — ниже нуля)
2	Запись и чтение	При установке в 0 используется режим генерации одного прерывания при нарушении заданных границ, иначе (1) прерывание генерируется постоянно, пока текущее значение температуры выходит за установленные пределы
3	Запись и чтение	Установка режима обработки температуры (0 — 8-разрядные преобразования, 0 — 11-разрядные преобразования)
4—7	Запись и чтение	Сюда записываются 4 младших бита для 11-разрядного положительного значения температуры, а для 8-разрядных значений температуры в бит 7 записывается знак плюса

Теперь осталось рассмотреть регистры данных, которые позволяют считывать и устанавливать параметры всех линий мониторинга оборудования. Список и назначение регистров данных для LM79 показаны в табл. 16.26, а для LM80 — в табл. 16.27. Линии 0—6 применяются для считывания напряжений блока питания компьютера. Обычно они распределены следующим образом: линия 0 — +2,5 В, линия 1 — +2,5 В, линия 2 — +3,3 В, линия 3 — +5 В, линия 4 — +12 В, линия 5 — –12 В и линия 6 — –5 В.

Таблица 16.26. Список регистров данных для LM79

Номер регистра	Назначение
20h	Чтение линии 0
21h	Чтение линии 1
22h	Чтение линии 2
23h	Чтение линии 3
24h	Чтение линии 4
25h	Чтение линии 5 (отрицательное напряжение)
26h	Чтение линии 6 (отрицательное напряжение)
27h	Чтение температуры
28h	Чтение текущего значения счетчика первого вентилятора

Таблица 16.26 (окончание)

Номер регистра	Назначение
29h	Чтение текущего значения счетчика второго вентилятора
2Ah	Чтение текущего значения счетчика третьего вентилятора
2Bh	Установка максимального значения для линии 0
2Ch	Установка минимального значения для линии 0
2Dh	Установка максимального значения для линии 1
2Eh	Установка минимального значения для линии 1
2Fh	Установка максимального значения для линии 2
30h	Установка минимального значения для линии 2
31h	Установка максимального значения для линии 3
32h	Установка минимального значения для линии 3
33h	Установка максимального значения для линии 4
34h	Установка минимального значения для линии 4
35h	Установка максимального значения для линии 5
36h	Установка минимального значения для линии 5
37h	Установка максимального значения для линии 6
38h	Установка минимального значения для линии 6
39h	Установка предельного значения температуры
3Ah	Установка минимального значения температуры
3Bh	Установка предельного значения счетчика первого вентилятора
3Ch	Установка предельного значения счетчика второго вентилятора
3Dh	Установка предельного значения счетчика третьего вентилятора
3Eh—3Fh	Резерв

Таблица 16.27. *Список регистров данных для LM80*

Номер регистра	Назначение
20h	Чтение линии 0
21h	Чтение линии 1
22h	Чтение линии 2

Таблица 16.27 *(окончание)*

Номер регистра	Назначение
23h	Чтение линии 3
24h	Чтение линии 4
25h	Чтение линии 5 (отрицательное напряжение)
26h	Чтение линии 6 (отрицательное напряжение)
27h	Чтение температуры
28h	Чтение текущего значения счетчика первого вентилятора
29h	Чтение текущего значения счетчика второго вентилятора
2Ah	Установка максимального значения для линии 0
2Bh	Установка минимального значения для линии 0
2Ch	Установка максимального значения для линии 1
2Dh	Установка минимального значения для линии 1
2Eh	Установка максимального значения для линии 2
2Fh	Установка минимального значения для линии 2
30h	Установка максимального значения для линии 3
31h	Установка минимального значения для линии 3
32h	Установка максимального значения для линии 4
33h	Установка минимального значения для линии 4
34h	Установка максимального значения для линии 5
35h	Установка минимального значения для линии 5
36h	Установка максимального значения для линии 6
37h	Установка минимального значения для линии 6
38h	Установка предельного значения температуры (8-разрядное значение)
39h	Установка минимального значения температуры (8-разрядное значение)
3Ah	Установка предельного значения температуры (12-разрядное значение)
3Bh	Установка минимального значения температуры (12-разрядное значение)
3Ch	Установка предельного значения счетчика первого вентилятора
3Dh	Установка предельного значения счетчика второго вентилятора
3Eh—3Fh	Резерв

Внешний датчик температуры может быть выполнен (в нашем контексте) на базе микросхем LM75, LM83, LM87, LM90, LM92 или LM99. Поскольку их параметры (чувствительность и время срабатывания) различаются, я не стану здесь рассматривать каждый из них, вы самостоятельно можете получить нужную информацию в Интернете. Я расскажу только о температурном датчике LM75. Он представляет собой отдельную микросхему для измерения температуры в диапазоне от −55 до +125 °С. Микросхема содержит общий управляющий регистр (чтение и запись), через который можно выбирать дополнительные регистры для чтения или записи. Формат общего регистра показан в табл. 16.28.

Таблица 16.28. *Формат общего регистра для LM75*

Биты	7	6	5	4	3	2	1	0
Описание	0	0	0	0	0	0	Выбор регистра	

Приведем некоторые комментарии к табл. 16.28.

- Биты 2—7 не используются и должны быть установлены в 0.
- Биты 0—1 определяют номер дополнительного регистра. Возможны следующие значения: `00b` — регистр температуры, `01b` — регистр конфигурации, `10b` — регистр предельного значения гистерезиса температуры, `11b` — регистр стандартного значения температуры.

Регистр температуры доступен только для чтения и имеет размер 16 бит. Используются только старшие 8 бит (7—15). Установка 0 бита в 1 определяет значение температуры в 0,5 °С.

Регистр конфигурации доступен для чтения и записи. Он имеет размер 8 бит. Формат регистра показан в табл. 16.29.

Таблица 16.29. *Формат регистра конфигурации для LM75*

Биты	7	6	5	4	3	2	1	0
Описание	0	0	0	Дефекты		Полярность	Режим	Питание

Приведем краткое описание таблицы.

- Бит 0, установленный в 1, позволяет выключить подачу напряжения к микросхеме LM75.
- Бит 1 определяет режим работы микросхемы: 1 — генерировать прерывания, 0 — режим сравнения значений.
- Бит 2 позволяет установить полярность измерений (1 — выше нуля, 0 — ниже нуля).

- Биты 3—4 позволяют установить количество обнаруженных дефектов перед выходом в рабочий режим, которые будут игнорироваться. Для завершения процесса инициализации микросхемы достаточно 100 мс.
- Биты 5—7 должны быть установлены в 0.

Регистры установки предельного значения гистерезиса температуры и стандартного значения температуры доступны для чтения и записи. Они имеют размер 16 бит, хотя используются только старшие 8 бит (7—15). По умолчанию значение предельного гистерезиса температуры равно 80 °С, а стандартное значение — 75 °С.

Вот и все основные сведения о температурном датчике. Теперь рассмотрим примеры программирования функций мониторинга системы. Вначале выполним сброс и последующую инициализацию микросхемы, как показано в листинге 16.1. Хочу заметить, что для считывания текущих значений температуры, напряжений и оборотов вентилятора выполнять инициализацию не нужно. Это делается только в том случае, если вы хотите перенастроить параметры микросхемы по-новому или когда микросхема не отвечает.

Листинг 16.1. Инициализация микросхемы мониторинга

```
Init_LM79 proc
; выполняем сброс микросхемы
xor AX, AX    ; обнуляем регистр
mov DX, 290h ; номер базового порта
; выбираем регистр конфигурации 40h
add DX, 5     ; регистр адреса x5h
mov AL, 40h   ; номер регистра конфигурации
out DX, AL    ; записываем значение в порт
; записываем значение инициализации в регистр данных x6h
inc DX        ; регистр данных x6h
mov AL, 80h   ; код инициализации
out DX, AL    ; записываем значение в порт
; проверяем наличие микросхемы мониторинга
dec DX, 1     ; регистр адреса x5h
mov AL, 40h   ; номер регистра конфигурации
out DX, AL    ; записываем значение в порт
; читаем значение регистра конфигурации 40h из порта данных x6h
inc DX        ; регистр данных x6h
in  AL, DX    ; читаем байт из порта
cmp AL, 08h   ; если результат не равен 08h,
jne NO_MONITOR; микросхема мониторинга отсутствует
; выполняем инициализацию микросхемы
dec DX        ; регистр адреса x5h
```

```
mov AL, 43h  ; регистр SMI 1
out DX, AL   ; записываем значение в порт
mov AL, 0FFh ; значение по умолчанию для регистра SMI 1
inc DX       ; регистр данных x6h
out DX, AL   ; записываем значение в порт
dec DX       ; регистр адреса x5h
mov AL, 44h  ; регистр SMI 2
out DX, AL   ; записываем значение в порт
mov AL, 0FFh ; значение по умолчанию для регистра SMI 2
inc DX       ; регистр данных x6h
out DX, AL   ; записываем значение в порт
dec DX       ; регистр адреса x5h
mov AL, 45h  ; регистр NMI 1
out DX, AL   ; записываем значение в порт
mov AL, 0FFh ; значение по умолчанию для регистра NMI 1
inc DX       ; регистр данных x6h
out DX, AL   ; записываем значение в порт
dec DX       ; регистр адреса x5h
mov AL, 46h  ; регистр NMI 2
out DX, AL   ; записываем значение в порт
mov AL, 0FFh ; значение по умолчанию для регистра NMI 2
inc DX       ; регистр данных x6h
out DX, AL   ; записываем значение в порт
; запускаем мониторинг системы
dec DX, 1    ; регистр адреса x5h
mov AL, 40h  ; номер регистра конфигурации
out DX, AL   ; записываем значение в порт
; читаем значение регистра конфигурации 40h из порта данных x6h
inc DX       ; регистр данных x6h
in  AL, DX   ; читаем байт из порта
or  AL, 01b  ; устанавливаем бит 1 для включения мониторинга
mov BL, 08h  ; копируем значение
not BL       ; инверсия
and AL, BL   ; разрешаем прерывания
or  AL, 04h  ; определяем генерацию NMI прерываний
mov BL, 20h  ; копируем значение
not BL       ; инверсия
and AL, BL   ; выбираем прерывания IRQ
mov AH, AL   ; временно сохраняем значение
; сохраняем новое значение байта конфигурации
dec DX, 1    ; регистр адреса x5h
mov AL, 40h  ; номер регистра конфигурации
out DX, AL   ; записываем значение в порт
```

```
; записываем значение инициализации в регистр данных x6h
inc DX       ; регистр данных x6h
mov AL, AH   ; байт конфигурации
out DX, AL   ; записываем значение в порт
ret
Init_LM79 endp
```

Аналогичный пример для C++ показан в листинге 16.2.

Листинг 16.2. Инициализация микросхемы мониторинга в C++

```
// пишем функцию инициализации микросхемы LM79
bool Init_LM79 ()
{
   unsigned int uBasePort = 0x290; // номер базового порта
   DWORD dwResult = 0;
   // выполняем сброс микросхемы
   // выбираем регистр конфигурации 40h
   outPort ( uBasePort + 5, 0x40, 1);
   // записываем значение инициализации в регистр данных x6h
   outPort ( uBasePort + 6, 0x80, 1);
   // проверяем наличие микросхемы мониторинга
   // выбираем регистр конфигурации 40h
   outPort ( uBasePort + 5, 0x40, 1);
   // читаем значение регистра конфигурации 40h из порта данных x6h
   inPort ( uBasePort + 6, &dwResult, 1);
   // если полученное значение не равно 0x08, микросхема отсутствует
   if ( dwResult != 0x08) return false;
   // выполняем инициализацию микросхемы
   // значение по умолчанию для регистра SMI 1
   // выбираем регистр SMI 1
   outPort ( uBasePort + 5, 0x43, 1);
   // записываем значение инициализации в регистр данных x6h
   outPort ( uBasePort + 6, 0xFF, 1);
   // выбираем регистр SMI 2
   outPort ( uBasePort + 5, 0x44, 1);
   // записываем значение инициализации в регистр данных x6h
   outPort ( uBasePort + 6, 0xFF, 1);
   // выбираем регистр NMI 1
   outPort ( uBasePort + 5, 0x45, 1);
   // записываем значение инициализации в регистр данных x6h
   outPort ( uBasePort + 6, 0xFF, 1);
   // выбираем регистр NMI 2
   outPort ( uBasePort + 5, 0x46, 1);
```

```
   // записываем значение инициализации в регистр данных x6h
   outPort ( uBasePort + 6, 0xFF, 1);
   // запускаем мониторинг системы
   // выбираем регистр конфигурации 40h
   outPort ( uBasePort + 5, 0x40, 1);
   // читаем значение регистра конфигурации 40h из порта данных x6h
   inPort ( uBasePort + 6, &dwResult, 1);
   // устанавливаем бит 1 для включения мониторинга
   dwResult |= 0x01;
   // разрешаем прерывания
   dwResult & = ~(0x08);
   // определяем генерацию NMI прерываний
   dwResult |= 0x04;
   // выбираем прерывания IRQ
   dwResult & = ~(0x20);
   // записываем значение в регистр конфигурации
   // выбираем регистр конфигурации 40h
   outPort ( uBasePort + 5, 0x40, 1);
   // записываем значение инициализации в регистр данных x6h
   outPort ( uBasePort + 6, dwResult, 1);
   return true; // выходим из функции
}
```

Для того чтобы остановить процесс мониторинга, можно применить код из листинга 16.3.

Листинг 16.3. Остановка процесса мониторинга

```
Stop_LM79 proc
xor AX, AX   ; обнуляем регистр
xor BX, BX   ; обнуляем регистр
mov DX, 290h ; номер базового порта
; выбираем регистр конфигурации 40h
add DX, 5    ; регистр адреса x5h
mov AL, 40h  ; номер регистра конфигурации
out DX, AL   ; записываем значение в порт
; читаем значение регистра конфигурации 40h из порта данных x6h
inc DX       ; регистр данных x6h
in  AL, DX   ; получаем байт из порта
mov BL, 01h  ; копируем значение для остановки мониторинга
not BL       ; инверсия
and AL, BL   ; бит 0 регистра конфигурации сбрасываем в 0
mov AH, AL   ; сохраняем значение
```

```
; сохраняем новое значение байта конфигурации
dec DX        ; регистр адреса x5h
mov AL, 40h   ; номер регистра конфигурации
out DX, AL    ; записываем значение в порт
inc DX        ; регистр данных x6h
mov AL, AH    ; копируем сохраненное значение
out DX, AL    ; записываем значение в порт
ret
Stop_LM79 endp
```

Аналогичный пример для C++ показан в листинге 16.4.

Листинг 16.4. Остановка процесса мониторинга в C++

```
// пишем функцию для остановки процесса мониторинга
void Stop_LM79 ()
{
   unsigned int uBasePort = 0x290; // номер базового порта
   DWORD dwResult = 0;
   // выбираем регистр конфигурации 40h
   outPort ( uBasePort + 5, 0x40, 1);
   // читаем значение регистра конфигурации 40h из порта данных x6h
   inPort ( uBasePort + 6, &dwResult, 1);
   // бит 0 регистра конфигурации сбрасываем в 0
   dwResult & = ~(0x01);
   // сохраняем новое значение байта конфигурации
   outPort ( uBasePort + 5, 0x40, 1);
   outPort ( uBasePort + 6, dwResult, 1);
}
```

А теперь рассмотрим пример, позволяющий получить текущее значение оборотов вентилятора (листинг 16.5).

Листинг 16.5. Получение текущего значения оборотов вентилятора

```
; переменная для хранения числа оборотов
Cooler_Count dw ?
; общий код программы
Get_CoolerSpeed proc
xor EAX, EAX ; обнуляем регистр
xor EBX, EBX ; обнуляем регистр
mov DX, 290h ; номер базового порта
; получаем текущее значение счетчика из регистра 28h
add DX, 5    ; регистр адреса x5h
```

```
mov AL, 28h  ; номер регистра счетчика для первого вентилятора
out DX, AL   ; записываем значение в порт
inc DX       ; регистр данных x6h
in  AL, DX   ; получаем байт из порта
; проверяем, крутится ли он вообще
cmp AL, 0FFh     ; если значение AL равно FFh,
je COOLER_STOPPED; вентилятор не крутится или сгорел
mov BL, AL   ; сохраняем значение
; получаем значение делителя из регистра 47h
dec DX       ; регистр адреса x5h
mov AL, 47h  ; номер регистра делителя
out DX, AL   ; записываем значение в порт
inc DX       ; регистр данных x6h
in  AL, DX   ; получаем байт из порта
; выделяем значение делителя для первого вентилятора
shr AL, 4    ; сдвигаем вправо на 4 разряда
and AL, 3    ; выделяем нужное
shl AL, 1    ; сдвигаем обратно на один разряд
; если значение счетчика равно 0, устанавливаем его в 1
cmp BL, 0    ; если 0,
jz COUNT_NULL; передаем управление для установки не нулевого значения
dalee:
; считаем обороты вентилятора
mul BL       ; умножаем AL на BL
mov BX, AX   ; копируем результат в BX
mov EAX, 149970h; заносим константное значение 1 350 000
div BX       ; делим
mov Cooler_Count, AX; сохраняем значение в переменную
ret
COUNT_NULL:
add BL, 1    ; увеличиваем до 1
jmp short dalee; возвращаемся назад
Get_CoolerSpeed endp
```

В листинге 16.6 показан аналогичный пример для C++.

Листинг 16.6. Получение текущего значения оборотов вентилятора в C++

```
// пишем функцию для определения числа оборотов для вентилятора
unsigned int Get_CoolerSpeed ( unsigned int uCooler)
{
   unsigned int uBasePort = 0x290; // номер базового порта
   unsigned int uCurrentCounter = 1; // счетчик
```

```
  unsigned int uCurrentDiv = 0; // делитель
  DWORD dwResult = 0;
  if ( ( uCooler < 1) || ( uCooler > 3)) return 0;
  // получаем текущее значение счетчика из регистра 28h
  outPort ( uBasePort + 5, 0x28, 1);
  // читаем значение регистра конфигурации 28h из порта данных x6h
  inPort ( uBasePort + 6, &dwResult, 1);
  // сохраняем его в переменную
  uCurrentCounter = dwResult;
  // проверяем, крутится ли вообще вентилятор
  if ( uCurrentCounter == 0xFF)
       return 0; // вентилятор не крутится или сгорел
  // получаем значение делителя из регистра 47h
  outPort ( uBasePort + 5, 0x47, 1);
  // читаем значение регистра делителя 47h из порта данных x6h
  inPort ( uBasePort + 6, &dwResult, 1);
  // выделяем значение делителя для указанного вентилятора
  switch ( uCooler)
  {
  case 1:
    uCurrentDiv = 1 << ( ( dwResult >> 4) & 0x03);
    break;
  case 2:
    uCurrentDiv = 1 << ( ( dwResult >> 6) & 0x03);
    break;
  case 3:
    uCurrentDiv = 2; // всегда постоянное значение
    break;
  }
  // считаем количество оборотов в минуту для вентилятора
  // и выходим из функции
  return ( 1 350 000 / ( uCurrentCounter * uCurrentDiv));
}
```

Для установки нового значения оборотов вентилятора можно применить код из листинга 16.7.

Листинг 16.7. Установка нового значения оборотов для первого вентилятора

```
; задаем делитель частоты равным 4
Div_Cooler db 2
; общий код программы
Set_CoolerSpeed proc
xor EAX, EAX ; обнуляем регистр
```

```
xor EBX, EBX ; обнуляем регистр
mov DX, 290h ; номер базового порта
; получаем текущее значение из регистра 47h
add DX, 5    ; регистр адреса x5h
mov AL, 47h  ; номер регистра делителя
out DX, AL   ; записываем значение в порт
inc DX       ; регистр данных x6h
in  AL, DX   ; получаем байт из порта
mov BL, 3    ; копируем значение для удобства расчетов
shl BL, 4    ; выделяем разряды
not BL       ; инверсия
and AL, BL   ; копируем в AL с преобразованием
mov BL, Div_Cooler; копируем делитель
shl BL, 4    ; сдвигаем влево
or  AL, BL   ; объединяем с AL
mov BL, AL   ; временно сохраняем
; записываем результат в регистр делителя
dec DX       ; регистр адреса x5h
mov AL, 47h  ; номер регистра делителя
out DX, AL   ; записываем значение в порт
inc DX       ; регистр данных x6h
mov AL, BL   ; копируем сохраненное значение
out DX, AL   ; записываем его в порт
ret
Set_CoolerSpeed endp
```

Аналогичный пример для C++ показан в листинге 16.8.

Листинг 16.8. Установка нового значения оборотов для первого вентилятора в C++

```
void Set_CoolerSpeed_4 ()
{
   unsigned int uBasePort = 0x290; // номер базового порта
   unsigned int uCurrentDiv = 2; // делитель
   DWORD dwResult = 0;
   // получаем текущее значение из регистра 47h
   outPort ( uBasePort + 5, 0x47, 1);
   // читаем значение регистра делителя 47h из порта данных x6h
   inPort ( uBasePort + 6, &dwResult, 1);
   // выделяем значение делителя для указанного вентилятора
   dwResult &= ~ ( 3 << 4);
   uCurrentDiv = uCurrentDiv << 4;
   dwResult |= uCurrentDiv;
```

```
   // записываем результат в регистр делителя
   outPort ( uBasePort + 5, 0x47, 1);
   // записываем новое значение в порт данных
   outPort ( uBasePort + 6, dwResult, 1);
}
```

В листинге 16.9 показан способ считывания текущего значения температуры материнской платы.

Листинг 16.9. Получение текущей температуры материнской платы

```
MB_TEMP db ?
; общий код программы
Get_MB_Temp proc
xor AX, AX    ; обнуляем регистр
mov DX, 290h ; номер базового порта
; получаем текущее значение из регистра 27h
add DX, 5     ; регистр адреса x5h
mov AL, 27h   ; номер регистра делителя
out DX, AL    ; записываем значение в порт
inc DX        ; регистр данных x6h
; небольшая пауза
mov CX, 5000 ; устанавливаем счетчик
xor BX, BX    ; обнуляем регистр
@pause:
add BX, CX    ; чисто символически
loop @pause   ; повторяем 5000 раз
in  AL, DX    ; получаем байт из порта
mov MB_TEMP, AL; копируем значение полученной температуры в переменную
ret
Get_MB_Temp endp
```

Аналогичный пример для C++ показан в листинге 16.10.

Листинг 16.10. Получение текущей температуры материнской платы в C++

```
int Get_MB_Temp ()
{
   unsigned int uBasePort = 0x290; // номер базового порта
   DWORD dwResult = 0;
   // получаем текущее значение из регистра 27h
   outPort ( uBasePort + 5, 0x27, 1);
```

```
    // читаем значение температуры из порта данных x6h
    inPort ( uBasePort + 6, &dwResult, 1);
    return dwResult;
}
// пример использования функции
int iTemperature = 0;
char sxText[30];
// получаем текущее значение температуры
iTemperature = Get_MB_Temp ();
sprintf ( szText, "Температура: %2d.00 ", iTemperature);
```

И последний пример для получения текущих напряжений питания показан в листинге 16.11.

Листинг 16.11. Получение текущих напряжений питания в С++

```
float Get_Volt ( unsigned int OffsetReg)
{
    unsigned int uBasePort = 0x290; // номер базового порта
    DWORD dwResult = 0;
    float fResult = 0.0000, fV_1, fV_2;
    // в зависимости от номера регистра, считаем базовое значение
    switch ( OffsetReg)
    {
    case 0: // регистр 20h для нулевой линии
    case 1: // регистр 21h для первой линии
    case 2: // регистр 22h для второй линии
      fV_1 = 0.00;
      fV_2 = 0.0;
      break;
    case 3: // регистр 23h для третьей линии
      fV_1 = 2.98;
      fV_2 = 5.0;

      break;
    case 4: // регистр 24h для четвертой линии
      fV_1 = 3.00;
      fV_2 = 12.0;
      break;
    case 5: // регистр 25h для пятой линии
      fV_1 = 3.00;
      fV_2 = -12.0;
      break;
```

```
    case 5: // регистр 26h для шестой линии
      fV_1 = 3.50;
      fV_2 = -5.0;
      break;
    }
    // получаем текущее значение из регистра
    outPort ( uBasePort + 5, OffsetReg, 1);
    // читаем значение напряжения из порта данных x6h
    inPort ( uBasePort + 6, &dwResult, 1);
    // вычисляем значение напряжения
    fResult = ( 0.016 * ( float) dwResult) - fV_1 + fV_2;
    return fResult;
)
// применим функцию на практике
char buffer[300];
float fCore, fn5, fp5, fn12, fp12, fc33, f33;
// получаем значения напряжений
fCore = Get_Volt ( 0);
fn5 = Get_Volt ( 6);
fp5 = Get_Volt ( 3);
fn12 = Get_Volt ( 5);
fp12 = Get_Volt ( 4);
fc33 = Get_Volt ( 1);
f33 = Get_Volt ( 2);
// форматируем значения в удобочитаемый вид
sprintf ( buffer, "Vcore: %2.2f V \п -5 V: %2.2f V \n \
+5 V: %2.2f V \n -12 V: %2.2f V \n +12 V: %2.2f V \n \
3,3 V: %2.2f V \n 3,3 V: %2.2f V \n", fn5, fp5, fn12, fp12, fc33, f33);
```

На этом можно завершить тему аппаратного мониторинга системы. Существует еще много различных модификаций датчиков, и обо всех рассказать просто невозможно. Однако материал этой главы поможет вам самостоятельно разобраться со всеми другими существующими модулями и сенсорами. Главное — не забывайте обращаться за обновленной информацией на сайты производителей оборудования.

ГЛАВА 17

Параллельный и последовательный порты

Последовательный и параллельный порты так прочно вошли в конфигурацию компьютера, что, несмотря на появление новых высокоскоростных интерфейсов, продолжают оставаться без изменений и дополнений. Почему так происходит и что в них такого особенного? Во-первых, это простота исполнения и отработанный годами протокол передачи данных. Во-вторых, эти интерфейсы идеально подходят для подключения определенных низкоскоростных устройств: модема и принтера. В-третьих, данные порты (особенно последовательный) широко применяются для работы со специфическим промышленным и научным оборудованием. В-четвертых, в мире накопилось так много устройств, использующих эти интерфейсы, что еще нескоро производители современных компьютерных систем откажутся от них.

В любом случае мы рассмотрим вопросы программирования параллельных и последовательных портов в следующем порядке:

1. С использованием функций BIOS.
2. С использованием аппаратных портов.
3. С помощью интерфейса Win32 API.

17.1. Общие сведения

Параллельная передача данных построена по принципу одновременной передачи 8 битов информации по 8 отдельным линиям (проводам). К тому же данные могут передаваться лишь в одном направлении. Для организации двунаправленной передачи потребуется специальный кабель и соответствующее программное обеспечение.

В процессе передачи данных (1 байт за один цикл) приемная и передающая стороны сообщают по специальной линии о своем состоянии. Для этого применяются специальные сигналы. Когда устройство (например, принтер) занято, оно выдает сигнал BUSY, а когда программа подготовила байт данных для передачи, она передает устройству сигнал STROBE. Существуют также специальные линии для контроля передаваемых данных, для сообщения об отсутствии бумаги, для передачи сообщений об ошибках и готовности устройства.

Параллельный интерфейс (LPT — Line Printer) в компьютере состоит из трех независимых портов: LPT1, LPT2 и LPT3. Кроме того, имеются три основных типа параллельных портов: стандартный, EPP (Enhanced Parallel Port) и ECP (Extended Capability Port). Стандартный порт поддерживает только однонаправленную передачу данных (от компьютера к устройству). Максимальная скорость передачи может составлять от 120 до 200 Кб/сек. Порт EPP является двунаправленным и поддерживает скорость передачи до 2 Мб/сек. Данный порт позволяет использовать канал прямого доступа к памяти (DMA). Порт ECP аналогичен EPP, но дополнительно позволяет сжимать данные, что еще больше увеличивает общую скорость обмена данными. Сжатие реализуется как программно, так и аппаратно.

Последовательный порт реализован на базе интерфейса RS-232. Как правило, компьютер поддерживает до четырех портов: COM1, COM2, COM3 и COM4. Передача данных выполняется побитно в двух направлениях и с одинаковой частотой. Передаваемый блок данных в асинхронном режиме состоит из стартового бита, битов данных (8) и стоп-бита. Скорость передачи данных измеряется в бодах (бит в секунду вместе со служебными битами) и может иметь следующие значения: 110, 300, 1 200, 2 400, 4 800, 9 600, 19 200, 38 400, 57 600 и 115 200. Если два устройства (модема) имеют разные скорости передачи, будет использована наименьшая из них. Стартовый и стоп-бит определяют начало и окончание блока данных. Дополнительно для выявления ошибок передачи добавляется бит четности. Его использование имеет три режима: без бита четности (No Parity), нечетный бит четности (Odd Parity) и четный бит четности (Even Parity). Перед началом передачи данных следует выполнить настройку и инициализацию обоих устройств.

17.2. Использование функций BIOS

17.2.1. Работа с параллельным портом

Поддержка параллельного порта BIOS выполняется с помощью прерывания int 17h. Существуют всего три стандартные функции, предназначенные для работы с этим портом. Рассмотрим их по порядку.

17.2.1.1. Функция *00h*

Эта функция позволяет передать на принтер один символ.

Использование:

1. В регистр AH следует поместить код функции 00h.
2. В регистр AL следует записать код символа (в том числе и управляющий).
3. В регистр DX записывается порядковый номер порта (0—2).
4. Вызывается прерывание BIOS int 17h.

Выход:

После выполнения функции в регистр AH будет записан результат выполнения операции в виде битовой маски (табл. 17.1). Как правило, установленные биты 4 и 5 сигнализируют о том, что принтер отсутствует на выбранном порту.

Таблица 17.1. Коды состояния для функций принтера

Бит	Описание
0	Тайм-аут или ошибка по времени (1 — ошибка, 0 — нет ошибки)
1—2	Зарезервированы и не используются
3	Ошибка ввода-вывода (1 — ошибка, 0 — нет ошибки)
4	Выбор принтера (1 — принтер доступен, 0 — принтер недоступен)
5	Закончилась бумага (1 — нет бумаги)
6	Подтверждение приема данных (0 — подтверждение есть)
7	Готовность принтера (1 — принтер не занят, 0 — принтер занят)

В реальных условиях обычно проверяется бит 3 (ошибка ввода-вывода). Пример вывода строки символов на принтер показан в листинге 17.1.

Листинг 17.1. Вывод строки символов на принтер

```
Message db 'Testing of the printer$'
mov DX, 0    ; выбираем первый принтер (LPT1)
mov CX, 23   ; количество символов, которые нужно напечатать
mov BX offset Message
@dalee:
mov AH, 0    ; код функции 00h
mov AL, [BX] ; первый символ
```

```
int 17h       ; вызываем прерывание
test AH, 01000b; проверяем бит ошибки
jnz ERROR_HND; вызываем обработчик ошибки
inc BX
loop @dalee   ; следующий символ
```

Перед началом работы нужно инициализировать принтер (функция 01h) и только поле этого начинать печать. Если произошла ошибка во время печати, следует повторно провести инициализацию устройства.

17.2.1.2. Функция *01h*

Функция позволяет инициализировать указанный порт принтера. Данная функция должна быть вызвана перед началом работы принтера и далее по ходу печати, если возникнут ошибки.

Использование:

1. В регистр AH следует поместить код функции 01h.
2. В DX записывается порядковый номер порта (0—2).
3. Вызывается прерывание BIOS int 17h.

Выход:

После выполнения функции в регистр AH будет записан результат выполнения операции в виде битовой маски (см. табл. 17.1). Пример инициализации порта LPT1 представлен в листинге 17.2.

Листинг 17.2. Инициализация принтера, подключенного к порту LPT1

```
mov AH, 1     ; код функции 01h
mov DX, 0     ; выбираем первый принтер (LPT1)
int 17h       ; вызываем прерывание
test AH, 01000b; проверяем бит ошибки
jnz ERROR_HND; вызываем обработчик ошибки
```

17.2.1.3. Функция *02h*

Данная функция позволяет получить информацию о текущем статусе принтера.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. В регистр DX записывается порядковый номер порта (0—2).
3. Вызывается прерывание BIOS int 17h.

Выход:

После выполнения функции в регистр AH будет записан результат выполнения операции в виде битовой маски (см. табл. 17.1).

Вот и все функции для работы с параллельным портом.

17.2.2. Работа с последовательным портом

Теперь рассмотрим функции BIOS и прерывание int 14h, поддерживающие программирование последовательного порта.

17.2.2.1. Функция *00h*

Эта функция позволяет инициализировать последовательный порт. Эту функцию следует всегда вызывать перед началом работы.

Использование:

1. В регистр AH следует поместить код функции 00h.
2. В AL следует записать байт инициализации. Формат этого байта показан в табл. 17.2.
3. В DX записывается порядковый номер последовательного порта (0—3).
4. Вызывается прерывание BIOS int 14h.

***Таблица 17.2.** Формат байта инициализации*

Бит	Описание
0—1	Размер данных (10b — 7 битов, 11b — 8 битов)
2	Количество стоповых битов (0 — один, 1 — два)
3—4	Бит четности (00b — нет, 01b — нечетный, 10b — нет, 11b — четный)
5—7	Скорость передачи (000b — 110 или 19 200 бод, 001b — 150 или 38 400 бод, 010b — 300 бод, 011b — 600 бод, 100b — 1200 бод, 101b — 2400 бод, 110b — 4800 бод и 111b — 9600 бод)

Выход:

После выполнения функции в регистр AH будет записано состояние порта (табл. 17.3), а в регистр AL — состояние модема (табл. 17.4).

***Таблица 17.3.** Формат байта состояния порта*

Бит	Описание
0	Готовность данных (1 — готовы, 0 — не готовы)
1	Ошибка переполнения данных (1 — есть, 0 — нет)

Таблица 17.3 (окончание)

Бит	Описание
2	Ошибка четности (1 — есть, 0 — нет)
3	Ошибка синхронизации данных (1 — есть, 0 — нет)
4	Обнаружен перерыв в передаче данных (1 — да, 0 — нет)
5	Регистр хранения данных пустой (1 — нет данных, 0 — есть данные)
6	Регистр сдвига пустой (1 — нет данных, 0 — есть данные)
7	Тайм-аут или ошибка по времени (1 — ошибка, 0 — нет ошибки)

***Таблица 17.4.** Формат байта состояния модема*

Бит	Описание
0	Изменилось состояние на входной линии CTS (Clear To Send): 1 — есть
1	Изменилось состояние на входной линии DSR (Data Set Ready): 1 — есть
2	Изменилось состояние на входной линии RI (Ring Indicator): 1 — есть
3	Изменилось состояние на входной линии DCD (Data Carrier Detect): 1 — есть
4	Произошел сброс для передачи на входной линии CTS (Clear To Send): 1 — сброс
5	Готовность модема для чтения данных на входной линии DSR (Data Set Ready): 1 — готов, 0 — не готов
6	Состояние индикатора звонка на входной линии RI (Ring Indicator): 1 — есть сигнал
7	Сигнал обнаружения несущей на входной линии DCD (Data Carrier Detect): 1 — обнаружен

Пример инициализации последовательного порта COM1 представлен в листинге 17.3.

Листинг 17.3. Инициализация последовательного порта COM1

```
Init_COM_1 proc
xor AX, AX   ; обнуляем регистр
mov AL, 10100011b; 4800 бод, без бита четности, 1 стоп-бит, 8 битов
mov DX, 0    ; порт COM1
int 14h      ; вызываем прерывание
ret
Init_COM_1 endp
```

17.2.2.2. Функция *01h*

Функция позволяет записать один символ в последовательный порт.

Использование:

1. В регистр AH следует поместить код функции 01h.
2. В AL следует записать код передаваемого символа.
3. В DX записывается порядковый номер последовательного порта (0—3).
4. Вызывается прерывание BIOS int 14h.

Выход:

После выполнения функции в регистр AH будет записано состояние порта (см. табл. 17.3). Простой пример использования данной функции показан в листинге 17.4.

Листинг 17.4. Передача введенного с клавиатуры символа в порт

```
; инициализируем порт COM2
call Init_COM_2
; проверяем нажатие любой клавиши
@repeat:
mov AH, 1     ; проверяем наличие символа в буфере клавиатуры
int 16h       ; вызываем прерывание
jz  @repeat   ; если нет, повторяем
mov AX, 0     ; читаем символ
int 16h       ; вызываем прерывание
test AL, AL   ; если получен расширенный код
jz  READ_EXT  ; обрабатываем его в другом месте
mov AH, 1     ; передаем символ в модем
mov DX, 1     ; порт COM2
int 14h       ; вызываем прерывание
```

17.2.2.3. Функция *02h*

Данная функция позволяет прочитать символ из последовательного порта.

Использование:

1. В регистр AH следует поместить код функции 02h.
2. В DX записывается порядковый номер последовательного порта (0—3).
3. Вызывается прерывание BIOS int 14h.

Выход:

После выполнения функции в регистр AH будет записано состояние порта (см. табл. 17.3), в AL — код прочитанного символа. Простой пример использования данной функции показан в листинге 17.5.

Листинг 17.5. Чтение данных из последовательного порта COM1

```
; инициализируем порт COM1
call Init_COM_1
mov AH, 2     ; читаем байт из порта
mov DX, 0     ; порт COM1
int 14h       ; вызываем прерывание
test AH, AH   ; если данные получены
jz OUT_SCREEN; выводим на дисплей
```

17.2.2.4. Функция *03h*

Функция позволяет получить текущее состояние порта и подключенного к нему модема.

Использование:

1. В регистр AH следует поместить код функции 03h.
2. В DX записывается порядковый номер последовательного порта (0—3).
3. Вызывается прерывание BIOS int 14h.

Выход:

После выполнения функции в регистр AH будет записано состояние порта (см. табл. 17.3), а в регистр AL — состояние модема (см. табл. 17.4). Пример использования данной функции представлен в листинге 17.6.

Листинг 17.6. Получение состояния последовательного порта

```
mov AH, 3     ; получаем состояние
mov DX, 2     ; порт COM2
int 14h       ; вызываем прерывание
test AH, 10000b; если есть перерыв в передаче данных
jnz ERROR_HND; передаем управление другому обработчику
test AH, 10b ; если обнаружена ошибка переполнения
jnz ERROR_HND; передаем управление другому обработчику
test AH, 01b ; если данные не готовы
jnz ERROR_HND; передаем управление другому обработчику
```

На этом можно завершить рассмотрение функций BIOS и перейти к непосредственному программированию портов.

17.3. Использование портов

Как мы уже знаем, стандартное количество параллельных портов ограничено тремя: LPT1, LPT2 и LPT3. Порты ввода-вывода распределены согласно табл. 17.5.

Таблица 17.5. Распределение параллельных портов ввода-вывода

LPT1 или LPT2	LPT2 или LPT3	Описание порта
378h	278h	Порт данных
379h	279h	Порт состояния
37Ah	27Ah	Порт управления

Можно напрямую обращаться к этим портам или же получить их значения из области памяти BIOS (0040h:0008h и 0040h:000Ah). Рассмотрим порты ввода-вывода подробнее.

Порт данных (378h или 278h) служит для передачи данных между компьютером и устройством (например, принтером) и доступен как для записи, так и для чтения.

Порт состояния (379h или 279h) позволяет получить информацию о текущем состоянии подключенного устройства и доступен только для чтения. Его формат представлен в табл. 17.6.

Таблица 17.6. Формат порта состояния

Бит	Описание
0—2	Не используются и должны быть установлены в 0
3	Ошибка ввода-вывода (1 — нет, 0 — есть)
4	Выбор принтера (1 — принтер доступен, 0 — принтер недоступен)
5	Закончилась бумага (1 — нет бумаги, 0 — есть)
6	Подтверждение приема данных (0 — подтверждение есть, 1 — принтер не готов)
7	Готовность принтера (1 — принтер не занят, 0 — принтер занят)

Порт управления (37Ah или 27Ah) позволяет инициализировать принтер и установить различные параметры работы. Формат этого порта показан в табл. 17.7.

Таблица 17.7. Формат порта управления

Бит	Описание
0	Состояние линии STROBE (1 — можно передать данные на принтер, 0 — стандартное состояние)

Таблица 17.7 (окончание)

Бит	Описание
1	Использование автоматического перевода строки LF (Ah) после возврата каретки CR (Dh): 1 — использовать
2	Инициализация порта принтера (0 — выполнить начальную инициализацию)
3	Выбор принтера (1 — принтер доступен, 0 — принтер недоступен)
4	Использовать прерывание от принтера (1 — использовать, 0 — нет)
5—7	Не используются и должны быть установлены в 0

А теперь рассмотрим практические примеры работы с параллельными портами. Перед началом работы необходимо инициализировать порт, как это сделано в листинге 17.7. Кроме того, инициализацию нужно выполнить повторно, если произошла ошибка передачи данных.

Листинг 17.7. Инициализация параллельного порта

```
mov DX, 37Ah ; порт управления LPT1 или LPT2
mov AL, 0Ch  ; подготавливаем порт к работе
out DX, AL   ; записываем значение в порт
; необходима небольшая задержка (не менее 0,5 мкс)
mov CX, 1500 ; начальное значение счетчика
@delay:
loop @delay  ; повтор
mov AL, 08h  ; инициализация порта
out DX, AL   ; записываем значение в порт
```

Аналогичный пример для C++ показан в листинге 17.8.

Листинг 17.8. Инициализация параллельного порта в C++

```
// пишем функцию инициализации
void InitLPT ( unsigned int Port)
{
   outPort ( Port, 0x0C, 1); // подготавливаем порт к работе
   Sleep ( 1); // небольшая задержка
   outPort ( Port, 0x08, 1); // инициализируем порт
}
// используем функцию для инициализации порта 27Ah
InitLPT ( 0x27A);
```

Для проверки готовности принтера можно использовать код в листинге 17.9.

Листинг 17.9. Проверка готовности порта LPT

```
xor AL, AL   ; обнуляем порт
mov DX, 379h ; порт состояния LPT1 или LPT2
in  AL, DX   ; читаем байт из порта
test AL, 10000b; проверяем 4 бит
jz  NO_ONLINE; принтер не подключен
```

Аналогичный пример для C++ показан в листинге 17.10.

Листинг 17.10. Проверка готовности порта LPT в C++

```
// пишем функцию проверки подключения принтера
bool IsReady_LPT ( unsigned int Port)
{
   DWORD dwResult = 0;
   inPort ( Port, &dwResult, 1); // читаем байт из порта
   if ( ( dwResult & 0x10) == 0x01)
     return true; // принтер подключен
   return false; // принтер не подключен
}
```

Для того чтобы послать на принтер байт данных, необходимо выполнить следующие действия:

1. Послать байт данных в порт данных.
2. Установить бит 0 управляющего порта в 0, а после небольшой паузы — в 1.
3. После этого следует ожидать готовности принтера к приему следующего байта данных. Для этого постоянно опрашивается бит 7 в регистре статуса. Когда он будет установлен в 1, можно переходить к пункту 2.

Пример записи данных в порт принтера показан в листинге 17.11.

Листинг 17.11. Передача данных в порт LPT

```
Message db 'Testing of the printer$'
; общий код программы
mov BX offset Message; указатель на строку символов
; проверяем готовность порта
mov CX, 5000 ; значение счетчика
mov DX, 379h ; порт состояния
```

```
@wait_LPT:
in  AL, DX   ; читаем байт из порта
test AL, 80h ; проверяем бит 7
loopne @wait_LPT; если бит 7 равен 1, повторяем
mov DX, 378h ; порт данных
@dalee:
mov AL. [BX] ; копируем символ в AL
out DX, AL   ; записываем значение в порт
mov DX, 37Ah ; порт управления
mov AL, 0Dh  ; устанавливаем сигнал STROBE
out DX, AL   ; записываем значение в порт
dec AL       ; сбрасываем сигнал STROBE
out DX, AL   ; записываем значение в порт
; проверяем готовность принтера к приему следующего символа
dec DX       ; порт состояния
@IsReady:
in  AL, DX   ; читаем байт из порта
test AL, 8   ; можем проверить ошибку
jnz ERROR_HND; передаем управление обработчику ошибок
test AL, 80h ; проверяем бит 7
jz  @IsReady ; если принтер не готов, повторяем
inc BX       ; следующий символ
dec DX       ; порт данных
```

Аналогичный пример для C++ показан в листинге 17.12.

Листинг 17.12. Передача данных в порт LPT для C++

```
// пишем функцию для передачи байта на принтер
int SetByte_LPT ( unsigned int BasePort, BYTE bData)
{
   DWORD dwResult  = 0;
   int iTimeWait = 5000;
   // проверяем готовность порта
   while ( -- iTimeWait > 0)
   {
       // читаем порт состояния
       inPort ( BasePort + 1, &dwResult, 1);
       if ( (dwResult & 0x80) == 0x00) break;
       // закончилось время ожидания
       if ( iTimeWait < 1) return 1; // вышло время ожидания
   }
   // записываем значение байта данных в порт
   outPort ( BasePort, bData, 1);
```

```
    // устанавливаем сигнал STROBE
    outPort ( BasePort + 2, 0x0D, 1);
    // сбрасываем сигнал STROBE
    outPort ( BasePort + 2, 0x0C, 1);
    // проверяем готовность принтера к приему следующего символа
    iTimeWait = 10000;
    while ( -- iTimeWait > 0)
    {
        // читаем порт состояния
        inPort ( BasePort + 1, &dwResult, 1);
        if ( (dwResult & 0x08) == 0x00) return 2; // произошла ошибка
        if ( (dwResult & 0x80) == 0x01) break;
        // закончилось время ожидания
        if ( iTimeWait < 1) return 1; // вышло время ожидания
    }
    return 0;
}
```

Работа с последовательным портом лишь немного сложнее параллельного. Стандартное распределение портов показано в табл. 17.8.

Таблица 17.8. *Распределения последовательных портов ввода-вывода*

Номер порта	Регистры ввода-вывода
COM1	3F8h—3FFh
COM2	2F8h—2FFh
COM3	3E8h—3EFh
COM4	2E8h—2Efh

Из таблицы видно, что для доступа к последовательному порту выделены 10 регистров, каждый из которых имеет свое назначение. Все регистры имеют размер 8 бит.

Регистр `3F8h` выполняет несколько задач и доступен для чтения и записи. Если бит 7 в регистре управления (`3FBh`) равен 0, то через `3F8h` можно читать и записывать данные. Если бит 7 в регистре управления (`3FBh`) равен 1, то регистр `3F8h` обрабатывает младший байт делителя частоты для скорости передачи данных.

Регистр `3F9h` также выполняет несколько задач и доступен для чтения и записи. Если бит 7 в регистре управления (`3FBh`) равен 0, то регистр `3F9h` управляет прерыванием (табл. 17.9). Если бит 7 в регистре управления (`3FBh`) равен 1, то `3F9h` обрабатывает старший байт делителя частоты для скорости передачи данных.

Таблица 17.9. *Формат регистра 3F9h в режиме управления прерыванием*

Бит	Описание
0	Состояние сигнала прерывания при наличии входных данных (1 — включен, 0 — выключен)
1	Состояние сигнала прерывания при опустошении выходного буфера (1 — включен, 0 — выключен)
2	Состояние сигнала прерывания при появлении ошибки или перерыва в передаче данных (1 — включен, 0 — выключен)
3	Состояние сигнала прерывания при изменении состояния модема (1 — включен, 0 — выключен)
4—7	Не используются и должны быть установлены в 0

Регистр 3FAh доступен для чтения и записи. Он позволяет определить причину прерывания в режиме чтения (табл. 17.10). В режиме записи регистр управляет режимом FIFO (табл. 17.11).

Таблица 17.10. *Формат регистра 3F9h в режиме чтения*

Бит	Описание
0	Состояние прерывания (1 — нет прерывания, 0 — есть прерывание)
1—2	Причина прерывания (00b — изменение состояния модема, 01b — опустошен буфер передачи, 10b — получены данные, 11b — произошла ошибка или перерыв в передаче данных)
3	Закончилось время ожидания (тайм-аут) для приемника FIFO
4—5	Не используются и должны быть установлены в 0
6—7	Наличие FIFO (11b — присутствует, 00b — отсутствует)

Таблица 17.11. *Формат регистра 3F9h в режиме записи*

Бит	Описание
0	Управление работой режима FIFO (1 — включен, 0 — выключен)
1	Очистка приемника FIFO (1 — очистить)
2	Очистка передатчика FIFO (1 — очистить)
3—5	Не используются и должны быть установлены в 0
6—7	Порог срабатывания для чтения данных (00b — 1 байт, 01b — 4 байта, 10b — 8 байт, 11b — 14 байт)

Регистр 3FBh доступен для чтения и записи. Он предназначен для управления линией связи. Формат регистра показан в табл. 17.12.

***Таблица 17.12.** Формат регистра управления*

Бит	Описание
0—1	Длина данных (00b — 5 бит, 01b — 6 бит, 10b — 7 бит, 11b — 8 бит)
2	Количество стоповых битов (0 — 1 бит, 1 — 2 бита)
3—5	Бит четности (000b — нет, 001b — нечетный, 011b — четный)
6	Состояние перерыва передачи, при котором порт выдает нули (1 — включить, 0 — выключить)
7	Управление делителем частоты для выбора скорости передачи данных (1 — установить значение делителя частоты через регистры 3F8h и 3F9h)

Регистр 3FCh доступен для чтения и записи. Он служит для управления работой модема. Формат регистра показан в табл. 17.13.

***Таблица 17.13.** Формат регистра управления модемом*

Бит	Описание
0	Управление линией DTR
1	Управление линией RTS
2	Управление линией OUT1 (0)
3	Управление линией OUT2 (1)
4	Самотестирование модема (1 — включить), при котором выход замыкается на вход
5—7	Не используются и должны быть установлены в 0

Регистр 3FDh доступен только для чтения. Он позволяет получить текущее состояние линии связи. Формат регистра был показан в табл. 17.3.

Регистр 3FEh доступен только для чтения. Он позволяет получить текущее состояние модема. Формат регистра был показан в табл. 17.4.

Регистр 3FCh доступен для чтения и записи. Он является резервным и не используется.

Перед началом работы нужно инициализировать последовательный порт. Для этого следует вначале установить бит 7 в 1 для регистра 3FBh, а затем записать желаемую скорость передачи (старший и младший байты) в регистры 3F8h и 3F9h. После этого в регистр 3FBh можно записать режим работы

линии, обнулив бит 7. Затем в регистр 3F9h надо записать 0, если прерывания не будут использованы, или установить биты 0—3 в соответствии с требуемыми прерываниями. Возможные значения для скорости передачи данных представлены в табл. 17.14.

Таблица 17.14. *Коды значений для установки скорости передачи данных*

Код значения		Скорость, бод
3F9h	3F8h	
04h	17h	110
01h	80h	300
00h	C0h	600
00h	60h	1 200
00h	30h	2 400
00h	20h	3 600
00h	18h	4 800
00h	10h	7 200
00h	0Ch	9 600
00h	06h	19 200
00h	03h	38 400
00h	02h	57 600
00h	01h	115 200

В листинге 17.13 показано, как выполняется инициализация последовательного порта без использования прерываний.

Листинг 17.13. Инициализация порта COM1

```
Init_COM1 proc
mov AL, 80h  ; устанавливаем бит 7 в 1
mov DX, 3FBh ; регистр управления линией
out DX, AL   ; записываем данные в порт
; устанавливаем скорость передачи данных 9 600 бод
mov DX, 3F8h ; порт для установки младшего байта делителя
mov AL, 00h  ; младший байт делителя
out DX, AL   ; записываем данные в порт
```

```
mov AL, 0Ch   ; старший байт делителя
mov DX, 3F9h ; порт для установки старшего байта делителя
out DX, AL    ; записываем данные в порт
; настраиваем регистр управления линией 3FBh
mov DX, 3FBh ; порт управления линией
mov AL, 00000011b; 8 бит, 1 стоповый бит, без четности
out DX, AL    ; записываем данные в порт
; настраиваем регистр управления модемом 3FCh
mov DX, 3FCh ; порт управления линией
mov AL, 00001011b; DTR, RTS, OUT1, OUT2
out DX, AL    ; записываем данные в порт
; настраиваем регистр управления прерываниями 3F9h
mov DX, 3F9h ; порт 3F9h
mov AL, 0     ; запрещаем все прерывания
out DX, AL    ; записываем данные в порт
ret
Init_COM1 endp
```

Аналогичный пример для C++ показан в листинге 17.14.

Листинг 17.14. Инициализация последовательного порта в C++

```
// пишем функцию инициализации последовательного порта
void Init_COM ( unsigned int BasePort)
{
   // устанавливаем бит 7 в 1 в регистре управления линией
   outPort ( BasePort + 3, 0x80, 1);
   // устанавливаем скорость передачи данных 9 600 бод
   outPort ( BasePort, 0x00, 1); // младший байт делителя
   outPort ( BasePort + 1, 0x0C, 1); // старший байт делителя
   // настраиваем регистр управления линией
   // 8 бит, 1 стоповый бит, без четности
   outPort ( BasePort + 3, 0x03, 1);
   // настраиваем регистр управления модемом
   // DTR, RTS, OUT1, OUT2
   outPort ( BasePort + 4, 0x0B, 1);
   // настраиваем регистр управления прерываниями
   outPort ( BasePort + 1, 0x00, 1); // запрещаем все прерывания
}
```

Для проверки текущего состояния линии можно использовать код, представленный в листинге 17.15.

Листинг 17.15. Проверка состояния последовательного порта COM2

```
; проверка состояния линии перед записью данных в порт
IsRead_Write_COM proc
xor AL, AL   ; обнуляем AL
mov DX, 3FDh ; порт состояния линии
in  AL, DX   ; читаем байт состояния из порта
test AL, 10h ; если бит 5 установлен в 1
jnz NO_WRITE ; передаем управление процедуре ожидания
ret          ; иначе выходим
IsRead_Write_COM endp
; проверка состояния линии перед чтением данных из порта
IsRead_Read_COM proc
xor AL, AL   ; обнуляем AL
mov DX, 3FDh ; порт состояния линии
in  AL, DX   ; читаем байт состояния из порта
test AL, 10b ; если бит 1 установлен в 1
jnz NO_READ  ; передаем управление процедуре чтения данных из порта
ret          ; иначе выходим
IsRead_Read_COM endp
```

Аналогичный пример обработки состояния для C++ показан в листинге 17.16.

Листинг 17.16. Проверка состояния последовательного порта в C++

```
// проверка состояния линии перед записью данных в порт
bool IsRead_Write_COM ( unsigned int BasePort)
{
   DWORD dwResult = 0;
   inPort ( BasePort + 5, &dwResult, 1);
   if ( ( dwResult & 0x10) == 0x01)
     return false; // линия занята
   return true;
}
// проверка состояния линии перед чтением данных из порта
bool IsRead_Read_COM ( unsigned int BasePort)
{
   DWORD dwResult = 0;
   inPort ( BasePort + 5, &dwResult, 1);
   if ( ( dwResult & 0x02) == 0x01)
     return false; // в линии остались непрочитанные данные
   return true;
}
```

Теперь, когда последовательный порт готов к работе, можно записать в него данные (листинг 17.17).

Листинг 17.17. Запись байта данных в последовательный порт COM1

```
; байт для данных
Data_Byte db 55
; общий код программы
; пишем процедуру для передачи байта данных в порт COM1
SendByte_COM proc
mov DX, 3F8h ; порт данных
mov AL, Data_Byte; копируем байт данных
out DX, AL   ; записываем байт в порт
ret
SendByte_COM endp
```

В рассмотренном примере опущены проверки состояния. Если вы используете прерывания, проверять состояние линии не обязательно, но в своем обработчике необходимо контролировать регистр состояния линии и модема.

Аналогичный пример посылки в порт байта для C++ показан в листинге 17.18.

Листинг 17.18. Запись байта данных в последовательный порт для C++

```
// пишем функцию для записи байта в последовательный порт
void SendByte_COM ( unsigned int BasePort, char* data)
{
   // записываем байт в порт
   outPort ( BasePort, ( DWORD) data, 1);
}
```

Пример чтения байта из последовательного порта представлен в листинге 17.19.

Листинг 17.19. Чтение байта данных из последовательного порта COM1

```
; буфер для данных
Data_Byte db ?
; общий код программы
; пишем процедуру для чтения байта данных из порта COM1
ReadByte_COM proc
mov DX, 3F8h ; порт данных
```

```
in  AL, DX   ; читаем байт из порта
mov Data_Byte, AL; копируем байт данных
ret
ReadByte_COM endp
```

Аналогичный пример чтения байта для C++ показан в листинге 17.20.

Листинг 17.20. Чтение байта данных из последовательного порта в C++

```
// пишем функцию для чтения байта из последовательного порта
BYTE ReadByte_COM ( unsigned int BasePort)
{
   DWORD dwResult = 0;
   // читаем байт из порта
   inPort ( BasePort, &dwResult, 1);
   return ( BYTE) dwResult;
}
```

На этом я хотел бы завершить программирование портов ввода-вывода и немного рассказать о поддержке последовательных и параллельных портов в интерфейсе Win32 API.

17.4. Использование Win32 API

Работа с LPT- и COM-портами в Windows упрощена до минимума. Чтобы инициализировать порт, достаточно вызвать функцию `CreateFile` с соответствующими аргументами. После этого можно писать и читать данные в порт посредством стандартных функций `WriteFile` и `ReadFile`. Кроме того, для настройки портов и последующего управления ими предназначен целый ряд дополнительных функций. Рассмотрим основные моменты программирования портов подробнее.

Перед началом работы необходимо открыть порт и настроить его на требуемый режим работы (для COM). В листинге 17.21 показано, как можно это сделать.

Листинг 17.21. Начальная инициализация последовательного порта COM1

```
#include <windows.h>
// объявляем структуру для конфигурации последовательного порта
DCB dcb;
ZeroMemory ( &dcb, sizeof ( DCB));
// дескриптор порта
HANDLE hCom_1 = NULL;
```

```
// пишем функцию инициализации порта COM1
bool Init_COM1 ()
{
   // открываем порт COM1 ( для COM2 "COM2", для LPT1 "LPT1" и т. д.)
   hCom_1 = CreateFile ( "COM1", GENERIC_READ | GENERIC_WRITE, 0, NULL,
                         OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);
   if ( hCom_1 == INVALID_HANDLE_VALUE} // если порт не удалось открыть
      returп false; // выходим из функции
   // если порт успешно открыт, получаем его состояние
   if ( !GetCommState ( hCom_1, &dcb))
   {
    // если не удалось получить статус порта, выходим из функции
    CloseHandle ( hCom_1);
    return false; // выходим из функции
   }
   // настраиваем параметры порта
   dcb.BaudRate  = CBR_19200; // скорость передачи 19 200 бод
   dcb.ByteSize = 8; // размер байта данных
   dcb.StopBits = ONESTOPBIT; // один стоповый бит
   dcb.Parity = NOPATITY; // контроля четности нет
   // сохраняем новые параметры конфигурации для порта
   if ( !SetCommState ( hCom_1, &dcb))
   {
    // если не удалось настроить порт, выходим из функции
    CloseHandle ( hCom_1);
    return false; // выходим из функции
   }
   return true;
}
```

Теперь, когда наш порт открыт и сконфигурирован для работы, нужно настроить обработку сигналов DSR и CTS. Пример кода для активизации уведомлений для последовательного порта показан в листинге 17.22.

Листинг 17.22. Настройка уведомляющих событий для сигналов DSR и CTS

```
// объявляем структуру для асинхронного ввода-вывода данных
// и помещаем ее в глобальную область видимости
OVERLAPPED over;
ZeroMemory ( &over, sizeof ( OVERLAPPED));
// пишем функцию настройки событий
bool SetEventCOM ()
```

```
{
   // настраиваем мониторинг за определенными событиями порта
   if ( !SetCommMask ( hCom_1, EV_DSR | EV_CTS))
   {
    // если не удалось настроить порт, выходим из функции
    CloseHandle ( hCom_1);
    return false; // выходим из функции
    }
   // создаем объект события
   over.hEvent = CreateEvent ( NULL, FALSE, FALSE, NULL);
   if ( !over.hEvent)
   {
    // если не удалось создать событие, выходим из функции
    CloseHandle ( hCom_1);
    return false; // выходим из функции
   }
   return true;
}
```

На данный момент вся начальная подготовка для работы с портом закончена. Осталось только написать функции чтения и записи данных и функцию управления для последовательного порта. Примеры функций чтения и записи показаны в листинге 17.23.

Листинг 17.23. Функции чтения и записи для работы с последовательным портом

```
// функция чтения одного байта данных
BYTE ReadByteCOM ()
{
   BYTE read = 0;
   DWORD dwByteRead = 0;
   do
   { // читаем байт из порта
     if ( !ReadFile ( hCom_1,  &read, sizeof ( BYTE), &dwByteRead,
                      NULL))
        return 0;
   }  while ( !dwByteRead);
return read; // возвращаем прочитанный байт
}
// функция чтения массива данных
DWORD ReadData_COM ( void* Data, unsigned int uNumBytes)
{
   DWORD dwBytesRead = 0;
```

```
   if ( !ReadFile ( hCom_1, Data, uNumBytes, &dwBytesRead, NULL))
      return dwBytesRead; // сколько удалось прочитать
   return dwBytesRead; // возвращаем полное число прочитанных байтов
}
// функция для записи одного байта
bool WriteByteCOM ( BYTE bByte)
{
   BYTE write = 0;
   DWORD dwByteWrite = 0;
   if ( !WriteFile ( hCom_1, &write, sizeof ( BYTE), &dwByteWrite,
                     NULL))
       return false;
   return true;
}
// функция для записи массива данных
DWORD WriteDataCOM ( void* Data, unsigned int uNumBytes)
{
   DWORD dwBytesWrite = 0;
   if ( !WriteFile ( hCom_1, Data, uNumBytes, &dwByteWrite,
                     NULL))
       return dwBytesWrite; // сколько удалось записать
   return dwBytesWrite; // возвращаем полное число записанных байтов
}
```

Теперь пишем общую функцию для работы с портом (листинг 17.24).

Листинг 17.24. Общая функция для работы с последовательным портом

```
// добавляем в глобальную область данных значение для выделения сигнала
DWORD dwSignal;
void GeneralCOM ()
{
   // проверяем сигнал в линии
   if ( WaitCommEvent ( hCom_1, &dwSignal, &over))
   {
     if ( dwSignal & EV_DSR) // данные готовы для чтения
     {
       // читаем байт из порта
       BYTE data = ReadByteCOM ();
       // сохраняем полученный байт куда-либо
     }
     if ( dwSignal & EV_CTS) // можно писать данные в порт
```

```
      {
        // передаем извне байт и пишем его в порт
        WriteByteCOM ( myByte);
      }
    }
}
// после завершения работы следует вызвать функцию очистки ресурсов
void CloseCOM ()
{
   if ( over.hEvent)
   {
     CloseHandle ( over.hEvent); // закрываем объект события
     over.hEvent = NULL;
   }
   if ( hCom_1)
   {
     CloseHandle ( hCom_1); // закрываем порт COM1
     hCom = NULL;
   }
}
```

Вот в принципе и все. Дополнительные возможности, предоставляемые интерфейсом Win32 API, вы сможете добавить уже самостоятельно.

ЧАСТЬ II

Общие методы программирования в Windows

ГЛАВА 18

Элементы управления

Практически ни одна современная программа сегодня не обходится без интерактивного графического интерфейса (GUI), который, в свою очередь, базируется на многообразии элементов управления, как стандартных, так и пользовательских. Именно по внешнему виду пользователи "встречают" и делают первые выводы о программе. Поэтому для разработчика вопрос оформления оболочки приложения должен стоять не на последнем месте. Особенно это важно при создании игровых и обучающих программ, энциклопедий и детских учебников. Существует даже отдельная профессия дизайнера программ, который занимается только внешним оформлением приложения, но поскольку наша книга посвящена в первую очередь программированию, мы рассмотрим вопросы создания и использования всевозможных элементов управления.

Как вы знаете, в оболочку Visual C++ (Visual C++ .NET) входит визуальный редактор ресурсов, с помощью которого и создается "лицо" будущей программы. К сожалению, его возможности довольно ограничены по сравнению, например, с Visual Basic или Visual C#. В нем полностью отсутствуют такие полезные мелочи, как управление шрифтом, цветом, загрузка картинок и динамическое изменение формы элемента управления. Все это приходится программировать вручную, на что уходит половина всего времени, затраченного на разработку программы. Этот момент, кстати, отпугивает многих начинающих программистов, предпочитающих более простой путь использования возможностей Visual Basic или Visual C#. Да и разработчики крупных проектов для создания интерфейса применяют упрошенные языки программирования по причине экономии времени. Однако есть и очень серьезный недостаток такого подхода, выраженный в серьезном увеличении дистрибутива программы. Поэтому выбор остается за разработчиком, а я постараюсь убедить вас в том, что в Visual C++ можно так же легко работать с элементами управления, как и в других языках. Поверьте мне на сло-

во, ни один самый мощный редактор ресурсов не дает и половины той широты и функциональности, которую предоставляет прямое программирование Win32 API.

18.1. Стандартные элементы управления

Сначала мы разберем стандартные, а затем дополнительные элементы управления. Для приводимых в книге примеров будут использованы как возможности программного интерфейса Win32 API, так и библиотеки классов от Microsoft (MFC).

К стандартным элементам управления относятся:

1. Кнопка (Button).
2. Раскрывающийся список (Combo Box).
3. Поле редактирования (Edit Box).
4. Список (List Box).
5. Поле редактирования с расширенными возможностями (Rich Edit).
6. Линейка прокрутки (Scroll Bar).
7. Статический элемент (Static Text).

Рассмотрим работу с этими элементами управления подробнее.

18.1.1. Кнопка

Кнопка является основным средством интерактивного взаимодействия с пользователем программы. Мы изучим основные моменты программирования кнопки: изменение шрифта, цвета и внешнего вида.

Для управления шрифтом нам понадобится структура `LOGFONT`, определяющая параметры шрифта, и функция создания нового шрифта `CreateFontIndirect`. В листинге 18.1 показан пример установки нового шрифта для кнопки.

Листинг 18.1. Изменение шрифта на кнопке

```
// пишем функцию замены шрифта
bool SetFontButton ( HWND hDlg, int iButtonID, int iHeight, char* font,
              bool Bold, bool Italic, bool Underline)
{
   LOGFONT lf;
   HFONT hFont = NULL; // дескриптор шрифта
   // обязательно обнуляем структуру перед использованием
   memset ( &lf, 0, sizeof ( LOGFONT));
```

```
  // высота шрифта
  if ( iHeight > 6)
    lf.lfHeight = iHeight;
  else
    lf.lfHeight = 8;
  // имя похожего шрифта
  if ( lstrlen ( font) > 4) lstrcpy ( lf.lfFaceName, font);
  // жирный шрифт
  if ( Bold)
    lf.lfWeight = FW_BOLD;
  else
    lf.lfWeight = FW_NORMAL;
  // наклон
  if ( Italic) lf. lfItalic = true;
  // подчеркивание
  if ( Underline)
    lf.lfUnderline = true;
  // создаем шрифт
  hFont = CreateFontIndirect ( &lf);
  if ( hFont == NULL) // не удалось создать новый шрифт
    return false;
  // устанавливаем новый шрифт
  SendDlgItemMessage ( hDlg, iButtonID, WM_SETFONT,
                       ( WPARAM) hFont, 0L);
  // освобождаем ресурсы системы
  DeleteObject ( hFont);
  return true;
}
// . . .
// применяем нашу функцию при инициализации диалогового окна
case WM_INITDIALOG:
{
  SetFontButton ( hWndDlg, IDC_MYBUTTON, 12, "Arial", true, false,
                  false);
}
return true;
```

Как видно из примера, мы использовали структуру LOGFONT для создания шрифта с помощью функции CreateFontIndirect. После этого мы послали сообщение WM_SETFONT для созданной в редакторе ресурсов кнопки (IDC_MYBUTTON). Функцию изменения шрифта можно вызывать в любом месте программы, когда требуется заменить шрифт. Структура LOGFONT имеет много дополнительных параметров, позволяющих создать собственный, ни

на что не похожий шрифт, но в большинстве случаев вполне хватает стандартных. Кроме того, при вызове функции SetFontButton после инициализации диалога желательно вызвать функцию обновления UpdateWindow для кнопки. Чтобы освободить ресурсы системы, следует вызвать функцию DeleteObject.

Для изменения шрифта средствами библиотеки MFC можно использовать код из листинга 18.2. Функция DeleteObject здесь не нужна, поскольку освобождением ресурсов занимается класс CFont.

Листинг 18.2. Изменение шрифта на кнопке в MFC

```
// определяем класс работы со шрифтами и структуру LOGFONT
CFont m_font;
LOGFONT lf;
memset ( &lf, 0, sizeof ( LOGFONT));
// заполняем структуру данными
lf.lfHeight = 14; // высота шрифта
lf.lfWeight = FW_BOLD; // жирный шрифт
lstrcpy(lf.lfFaceName, "Garamond"); // имя похожего шрифта
// создаем новый шрифт
m_font.CreateFontIndirect ( &lf);
// устанавливаем созданный шрифт для кнопки IDC_MYBUTTON
GetDlgItem ( IDC_MYBUTTON)->SetFont ( &m_font);
// или таким способом, где m_Button указывает на класс CButton
m_Button.SetFont ( &m_font, true);
```

Однако чтобы изменить цвет выводимого шрифта, нам придется воспользоваться другим способом. В листинге 18.3 показан пример кода для работы с цветом.

Листинг 18.3. Изменение цвета текста на кнопке

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
   // создаем дескриптор кисти
   static HBRUSH hBrush;
   // получаем дескриптор окна нашей кнопки
   HWND hMyButton = GetDlgItem ( hwndDlg, IDC_MYBUTTON);
   // обработка сообщений
   switch ( message)
    {
      case WM_INITDIALOG: // инициализация диалога
```

```
        {
          // создаем логическую кисть с системным цветом кнопки
          hBrush = CreateSolidBrush ( GetSysColor ( COLOR_BTNFACE));
        }
        return true;
        case WM_CTLCOLORBTN: // обработка цвета для кнопки
        {
          // выделяем сообщение для нашей кнопки
          if ( ( HWND) lParam == hMyButton)
          {
           // устанавливаем желаемый цвет текста и ( или) фона кнопки
           // цвет текста на кнопке, например синий
           SetTextColor ( ( HDC) wParam, RGB ( 53, 57, 217));
           // цвет фона текста
           SetBkColor ( ( HDC) wParam, GetSysColor ( COLOR_BTNFACE));
           return ( LRESULT) hBrush;
          }
        }
        return true;
        case WM_DESTROY: // завершение работы программы
        {
          // удаляем логическую кисть
          DeleteObject ( hBrush);
        }
        return true;
    }
   return FALSE;
}
```

Итак, чтобы изменить цвет текста на кнопке, в первую очередь следует создать логическую кисть с помощью функции `CreateSolidBrush`. В качестве начального цвета можно задавать абсолютно любой, но мы применили стандартный системный цвет кнопок (`COLOR_BTNFACE`). Для изменения цвета текста или фона кнопки требуется обработать системное сообщение `WM_CTLCOLORBTN`. При этом параметр `wParam` указывает на дескриптор контекста (`HDC`), а `lParam` — на дескриптор окна кнопки. Когда для нашей кнопки приходит сообщение `WM_CTLCOLORBTN`, используем функцию `SetTextColor` для установки цвета текста, а если нужно, то вызываем и функцию `SetBkColor` для изменения цвета фона текста. Если вам необходимо использовать несколько цветов, на каждый следует создавать отдельную логическую кисть. После завершения работы с программой обязательно следует удалить созданную кисть, вызвав функцию `DeleteObject`. Макрос `RGB` позволяет определить каждую из трех составляющих цвета: красную, синюю и зеленую.

В библиотеке MFC изменение цвета реализуется не намного сложнее. В листинге 18.4 представлены отрывки из класса `MyClass` диалогового окна (название класса может быть любым).

Листинг 18.4. Изменение цвета текста на кнопке в MFC

```
// в объявлении класса ( MyClass.h) определяем указатель на класс CBrush
CBrush* m_pbrButton;
// в реализации конструктора ( MyClass.cpp) создаем логическую кисть
//{{AFX_DATA_INIT(InfoSys)
      m_pbrButton = new Cbrush ( HS_HORIZONTAL,
                                  GetSysColor ( COLOR_MENU));
//}}AFX_DATA_INIT
// добавляем в класс обработку сообщения WM_CTLCOLOR
// пишем реализацию для этого сообщения
HBRUSH MyClass :: OnCtlColor (CDC* pDC, CWnd* pWnd, UINT nCtlColor)
{
   HBRUSH hbr = Cdialog :: OnCtlColor ( pDC, pWnd, nCtlColor);
   switch ( nCtlColor)
   {
    case CTLCOLOR_BTN:
    {
       // проверяем дескриптор нашей кнопки
       if ( pWnd->GetDlgCtrlID () == IDC_MYBUTTON)
       {
         // устанавливаем желаемый цвет
         pDC->SetTextColor ( RGB ( 17, 12, 209));
         // если нужно, то и цвет фона текста
         pDC->SetBkColor ( RGB ( 130, 100, 209));
         // возвращаем кисть системе
         return ( HBRUSH) ( m_pbrButton->GetSafeHandle ());
       }
    }
     break;
    default:
     return hbr;
   }
}
// после завершения работы удаляем кисть
delete m_pbrButton;
```

Вот и все премудрости. Теперь рассмотрим более сложную задачу, которая заключается в изменении цвета кнопки. Чтобы ее решить, нам понадобится

полностью взять на себя рисование и отображение кнопки. Для этого, в первую очередь, необходимо в редакторе ресурсов установить флажок **Owner draw**, расположенный на вкладке **Styles** свойств кнопки. Затем следует добавить в главную функцию диалогового окна обработку системного сообщения `WM_DRAWITEM` и подготовить несколько дополнительных функций для рисования кнопки. В листинге 18.5 показано, как это делается.

Листинг 18.5. Изменение цвета кнопки

```
// отрывок главной функции диалогового окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
   // указатель на структуру поддержки рисования
   LPDRAWITEMSTRUCT lpDraw;
   // получаем дескриптор окна нашей кнопки
   HWND hMyButton = GetDlgItem ( hwndDlg, IDC_MYBUTTON);
   // обработка сообщений
   switch ( message)
    {
     case WM_DRAWITEM:
     {
       // выделяем информацию для рисования элемента управления
       lpDraw = ( LPDRAWITEMSTRUCT) lParam;
       // проверяем, кому передано сообщение
       if ( hMyButton == lpDIS->hwndItem) // если нам
       {
         // рисуем кнопку
         UINT uState = lpDIS->itemState; // текущее состояние кнопки
         // сохраняем размеры кнопки
         RECT SizeFocus, SizeBtn;
         CopyRect ( &SizeFocus, &lpDraw->rcItem);
         CopyRect ( &SizeBtn, &lpDraw->rcItem);
         // сохраняем текст кнопки
         char BtnText[100];
         GetWindowText ( hMyButton, BtnText, 100);
         // вычисляем размер прямоугольника фокуса
         SizeFocus.top += 4;
         SizeFocus.left += 4;
         SizeFocus.right -= 4;
         SizeFocus.bottom -= 4;
         // рисуем кнопку и выводим на нее сохраненный текст
         FillRectButton ( lpDIS->hDC, SizeBtn, RGB ( 255, 255, 0));
```

```
        DrawButton ( lpDIS->hDC, SizeBtn, 2);
        SetButtonText (lpDIS->hDC, SizeBtn, BtnText, RGB ( 0, 0, 0));
        // обрабатываем различные состояния кнопки
        if ( uState & ODS_FOCUS) // кнопка в фокусе
        {
          DrawFocusRect (lpDIS->hDC, &SizeFocus);
          // если кнопка выбрана
          if ( uState & ODS_SELECTED)
          {
            FillRectButton ( lpDIS->hDC, SizeBtn,
                             RGB ( 255, 255, 0));
            DrawButton ( lpDIS->hDC, SizeBtn, 2);
            SetButtonText ( lpDIS->hDC, SizeBtn, BtnText,
                            RGB ( 0, 0, 0));
            DrawFocusRect ( lpDIS->hDC, &SizeFocus);
          }
        }
        else if ( uState & ODS_DISABLED) // кнопка не активна
        {
          // обрабатываем, если нужно
        }
      }
    }
    break;
  }
  return FALSE;
}
```

Итак, мы получаем сообщение `WM_DRAWITEM` для нашей кнопки. После этого сохраняем текущее состояние кнопки (нажата, в фокусе и т. п.) и ее размеры. Далее с помощью дополнительных функций рисуем и выводим кнопку на экран. Теперь остается только обработать различные состояния кнопки, и задача решена. В данном примере обрабатывается только момент нажатия и получение фокуса нашей кнопки. В листинге 18.6 представлена реализация функций рисования.

Листинг 18.6. Функции рисования стандартной кнопки

```
// функция заполнения указанного прямоугольника цветом
void FillRectButton ( HDC hDC, RECT rect, COLORREF clr)
{
  // создаем сплошную кисть указанного цвета
  HBRUSH hBrush = CreateSolidBrush ( clr);
```

```
    // заполняем прямоугольник выбранным цветом кисти
    FillRect ( hDC, &rect, hBrush);
}
// функция для создания кнопки
void DrawButton ( HDC hDC, RECT rect, int Border)
{
    int i, pal, sizeBrd;
    COLORREF tmp, bgr, clr1, clr2;
    // определяем знак рамки
    if ( Border < 0)
       sizeBrd = -Border; // внутрь
    else
       sizeBrd = Border; // наружу
    // рисуем кнопку
    for (i = 0; i < sizeBrd; i += 1)
    {
      // создаем палитру из трех основных цветов
      pal = 255 / ( i + 2);
      clr1 = PALETTERGB ( pal, pal, pal);
      pal = 192 + ( 63 / ( i + 1));
      clr2 = PALETTERGB ( pal, pal, pal);
      // если размер рамки равен 1
      if ( sizeBrd == 1)
      {
        // устанавливаем стандартные цвета
        clr2 = RGB ( 255, 255, 255);
        clr1 = RGB ( 128, 128, 128);
      }
      // если размер рамки отрицательный, меняем цвета местами
      if ( sizeBrd < 0)
      {
        tmp = clr1;
        bgr = clr2;
      }
      else
      {
        tmp = clr2;
        bgr = clr1;
      }
      // рисуем две линии
      DrawL ( hDC, rect.left, rect.top, rect.right, rect.top, tmp);
      DrawL ( hDC, rect.left, rect.top, rect.left, rect.bottom, tmp);
```

```
      // обрабатываем промежуточное значение рамки
      if ( ( Border < 0) && ( i == sizeBrd - 1) && ( sizeBrd > 1))
      {
         DrawL ( hDC, rect.left + 1, rect.bottom - 1, rect.right,
                     rect.bottom - 1, RGB ( 1, 1, 1));
         DrawL ( hDC, rect.right - 1, rect.top + 1, rect.right - 1,
                     rect.bottom, RGB ( 1, 1, 1));
      }
      else
      (
         DrawL ( hDC, rect.left + 1, rect.bottom - 1, rect.right,
                     rect.bottom - 1, bgr);
         DrawL ( hDC, rect.right - 1, rect.top + 1, rect.right - 1,
                     rect.bottom, bgr);
      }
      // немного уменьшаем размер прямоугольника
      InflateRect ( &rect, -1, -1);
   }
}
// функция для рисования линии
void DrawL ( HDC hDC, long l, long t, long r, long b, COLORREF clr)
{
   // используем два пера
   HPEN newPen, oldPen;
   POINT point;
   // создаем новое перо
   newPen = CreatePen ( PS_SOLID, 1, clr);
   // выбираем новое перо и сохраняем старое
   oldPen = ( HPEN) SelectObject ( hDC, newPen);
   // новая позиция
   MoveToEx ( hDC, l, t, &point);
   // рисуем линию
   LineTo ( hDC, r, b);
   // восстанавливаем старое перо
   SelectObject ( hDC, oldPen);
   // освобождаем ресурсы
   DeleteObject ( newPen);
}
// функция для вывода текста на кнопку
void SetButtonText (HDC hDC, RECT rect, char* Caption, COLORREF TextClr)
{
   // устанавливаем новый и сохраняем старый цвет надписи
   COLORREF oldClr = SetTextColor ( hDC, TextClr);
```

```
    // устанавливаем режим прозрачности и отсечения фона
    SetBkMode ( hDC, TRANSPARENT);
    // выводим текст в область кнопки
    DrawText ( hDC, Caption, strlen ( Caption), &rect,
                DT_CENTER | DT_SINGLELINE | DT_VCENTER);
    // восстанавливаем прежний цвет надписи
    SetTextColor ( hDC, oldClr);
}
```

Теперь у нас есть такие же возможности для управления цветом, как и в языках Visual Basic или Visual C#, но при этом мы сохранили все достоинства прямого программирования интерфейса Win32 API.

18.1.2. Раскрывающийся список

Раскрывающийся список очень удобно использовать для хранения большого количества строковых значений с последующим выбором. Главным достоинством этого элемента является небольшой размер, который он занимает в окне программы.

Прежде чем перейти к программированию, мне бы хотелось объяснить один важный момент при использовании списка, о котором многие начинающие программисты не догадываются. Это касается создания данного элемента управления в редакторе ресурсов. Если вы заметили, созданный по умолчанию список не разворачивается при нажатии на кнопку со стрелкой. Чтобы решить данную проблему, следует щелкнуть левой кнопкой мыши прямо на стрелке. После этого просто потяните курсором за нижнюю сторону разметочной линии, пока не получите необходимый размер.

Теперь поговорим о том, как можно изменить цвет фона и текста для раскрывающегося списка. В отличие от кнопки, список состоит из двух элементов управления: поля редактирования (Edit Box) и стандартного списка (List Box). Поэтому для него требуется несколько иной подход. Посмотрите в листинге 18.7, как можно реализовать выбор цвета для раскрывающегося списка.

Листинг 18.7. Обработка цвета для раскрывающегося списка

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
    // создаем дескриптор кисти
    static HBRUSH hBrush;
    // получаем дескриптор окна нашего списка
    HWND hMyComboBox = GetDlgItem ( hwndDlg, IDC_MYCOMBOBOX);
```

```
   // обработка сообщений
   switch ( message)
    {
       case WM_INITDIALOG: // инициализация диалога
       {
         // создаем логическую кисть желтого цвета
         hBrush = CreateSolidBrush ( RGB ( 255, 255, 0));
       }
       return true;
       // обработка цвета для списка и окна редактирования
    case WM_CTLCOLORLISTBOX:
    case WM_CTLCOLOREDIT:
       {
          // задаем прозрачный фон
          SetBkMode ( ( HDC) wParam, TRANSPARENT);
          // цвет текста в списке, если необходимо
          SetTextColor ( ( HDC) wParam, RGB ( 200, 200, 27));
          // цвет фона текста тоже желтый
          SetBkColor ( ( HDC) wParam, RGB ( 255, 255, 0));
          return ( LRESULT) hBrush;
       }
       return true;
       case WM_DESTROY: // завершение работы программы
       {
         // удаляем логическую кисть
         DeleteObject ( hBrush);
       }
       return true;
    }
   return FALSE;
}
```

Вы можете комбинировать разные варианты для фона и текста. Если цвет текста не указан, будет использован системный цвет (как правило, это черный).

Работа со шрифтами ничем не отличается от работы с кнопкой, описанной ранее. Единственное, что нужно учитывать при изменении высоты шрифта, — это высота элементов списка. Ее следует изменять (при рисовании собственного списка) в соответствии с размером шрифта. Для этого используется сообщение CB_SETITEMHEIGHT.

А теперь давайте рассмотрим еще одну интересную и полезную возможность, позволяющую перетаскивать в раскрывающийся список файлы из

других окон посредством технологии "drag-and-drop" (Перетащить и отпустить). Прежде всего, построим специальный класс для поддержки процедуры перетаскивания файлов. В листингах 18.8 и 18.9 представлен образец такого класса.

Листинг 18.8. Файл DragDrop.h

```
// подключаем файл определений для оболочки
#include <shellapi.h>
// объявление класса
class DragDrop
{
public:
       DragDrop (); // конструктор
       ~DragDrop (); // деструктор
// общедоступные функции
       // выбор элемента управления
       void AttachControl ( HWND hControl);
       // обработка файлов
       void AddFile ( HWND hControl, const char* File);
private:
       HWND m_hwnd; // дескриптор окна элемента управления
       WNDPROC m_func; // указатель на функцию окна
       // связь класса с оконной функцией
       static LRESULT CALLBACK FilesDrop ( HWND hWnd, UINT msg,
                                   WPARAM wParam, LPARAM lParam);
       // обработка событий перетаскивания файлов для оконной функции
       long AddFilesProc ( HWND hWnd, UINT msg, WPARAM wParam,
                               LPARAM lParam);
}; // окончание класса
```

Листинг 18.9. Файл DragDrop.cpp

```
#include "stdafx.h"
#include "DrapDrop.h"
// реализация класса
DragDrop :: DragDrop ()
{
   m_hwnd = NULL;
   m_func = NULL;
}
```

```
DragDrop :: ~DragDrop ()
{
   // освобождаем ресурсы
   if ( m_hwnd != 0)
   {
     // восстанавливаем прежнюю процедуру окна
     SetWindowLong ( m_hwnd, GWL_WNDPROC, ( LONG) m_func);
     // удаляем точку входа для использованной процедуры окна
     RemoveProp ( m_hwnd, "DRAG_DROP_H_");
     m_hwnd = NULL;
   }
}
LRESULT CALLBACK DragDrop :: FilesDrop ( HWND hWnd, UINT msg,
                                 WPARAM wParam, LPARAM lParam)
{
   // получаем дескриптор из внешнего окна
   DragDrop* drag = ( DragDrop*) GetProp ( hWnd, "DRAG_DROP_H_");
   if ( !drag)return 0; // проверяем результат
   // подключаем нашу процедуру окна
   return drag->AddFilesProc ( hWnd, msg, wParam, lParam);
}
long DragDrop :: AddFilesProc ( HWND hWnd, UINT msg, WPARAM wParam,
                                 LPARAM lParam)
{
   // обрабатываем нужные нам сообщения
   switch ( msg)
   {
     case WM_CLOSE:
     case WM_DESTROY:
     {
       // при завершении работы освобождаем ресурсы
       SetWindowLong ( hWnd, GWL_WNDPROC, ( LONG) m_func);
       RemoveProp ( hWnd, "DRAG_DROP_H_");
       m_hwnd = 0;
      }
       break;
       // обрабатываем перетаскивание файлов
      case WM_DROPFILES:
       // определяем число перетаскиваемых файлов
       UINT uCountFiles = DragQueryFile ( ( HDROP) wParam,
                                    0xFFFFFFFF, 0, 0);
       // выделяем буфер для временного хранения имени файла
       char tmp[MAX_PATH];
```

```
        // обрабатываем файлы
        for ( unsigned int i = 0; i < uCountFiles; i++)
        {
          // получаем имя перетаскиваемого файла
          DragQueryFile ( ( HDROP) wParam, i, tmp, MAX_PATH);
          // пропускаем, если это папка
          if ( FILE_ATTRIBUTE_DIRECTORY != ( GetFileAttributes ( tmp) &
                                            FILE_ATTRIBUTE_DIRECTORY))
          {
             // добавляем имя файла в список
             AddFile ( hWnd, tmp);
          }
         }
         // завершаем процедуру перетаскивания
         DragFinish ( ( HDROP) wParam);
         return 0;
   } // end switch
   // передаем данные во внешнюю оконную процедуру
   return CallWindowProc ( m_func, hWnd, msg, wParam, lParam);
}
// функция инициализации и выбора элемента управления
void DragDrop :: AttachControl ( HWND hControl)
{
   // сохраняем дескриптор элемента управления
   m_hwnd = hControl;
   // устанавливаем на всякий случай поддержку Drag & Drop
   SetWindowLong ( m_hwnd, GWL_EXSTYLE, GetWindowLong ( m_hwnd,
                          GWL_EXSTYLE) | WS_EX_ACCEPTFILES);
   // закрепляем новое свойство для окна элемента управления
   SetProp ( m_hwnd, "DRAG_DROP_H_", ( HANDLE) this);
   // устанавливаем новую оконную функцию и сохраняем старую
   m_func = ( WNDPROC)SetWindowLong ( m_hwnd, GWL_WNDPROC,
                                     ( LONG) FilesDrop);
}
// функция обработки полученных файлов
void DragDrop :: AddFile ( HWND hControl, const char *File)
{
   // копируем в список только имена файлов
   char* s;
   s = strrchr ( File, '\\');
   char tmp[MAX_PATH];
   strcpy ( tmp, s+2);
```

```
    // добавляем имя файла в раскрывающийся список
    SendMessage ( hControl, CB_ADDSTRING, 0, ( LPARAM) ( LPCTSTR) tmp);
    SendMessage ( hControl, CB_SETCURSEL, 0, 0);
}
```

Получился удобный класс обработки перетаскивания файлов в окно списка. Для того чтобы применить класс `DragDrop`, необходимо выполнить две основные операции:

1. Объявить класс `DragDrop`.
2. Вызвать функцию `AttachControl`, передав ей в качестве аргумента дескриптор раскрывающегося списка.

В листинге 18.10 показан пример работы с классом `DragDrop`.

Листинг 18.10. Пример использования класса DragDrop

```
// подключаем необходимый файл
#include "DrapDrop.h"
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
    // объявляем наш класс
    static DragDrop dragCombo;
    // получаем дескриптор окна нашего списка
    HWND hMyComboBox = GetDlgItem ( hwndDlg, IDC_MYCOMBOBOX);
    // обработка сообщений
    switch ( message)
    {
        case WM_INITDIALOG: // инициализация диалога
        {
          // инициализируем класс и закрепляем за ним список
          drag.AttachControl ( hMyComboBox);
        }
        return true;
    }
    return false;
)
```

Вот и все, что необходимо сделать. Теперь можно смело копировать файлы в список посредством технологии "drag-and-drop". Думаю, потенциальные пользователи вашей программы быстро оценят простоту и удобство этого способа работы с файлами, ставшего поистине важным и необходимым во всех операционных системах Windows.

18.1.3. Поле редактирования

Этот простой элемент управления позволяет вводить и редактировать текст, а также взаимодействовать с буфером обмена. Мы, как и прежде, рассмотрим только те возможности, которые недоступны через редактор ресурсов.

Чтобы установить цвет фона или текста, можно применить код из листинга 18.11.

Листинг 18.11. Использование цвета для поля редактирования

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
   // создаем дескриптор кисти
   static HBRUSH hBrush;
   // получаем дескриптор поля нашего списка
   HWND hMyEditBox = GetDlgItem ( hwndDlg, IDC_MYEDITBOX);
   // обработка сообщений
   switch ( message)
    {
       case WM_INITDIALOG: // инициализация диалога
       {
         // создаем логическую кисть синего цвета
         hBrush = CreateSolidBrush ( RGB ( 0, 128, 255));
       }
       return true;
       // обработка цвета для поля редактирования
       case WM_CTLCOLOREDIT:
       {
         // выделяем сообщение для поля редактирования
         if ( ( HWND) lParam == hMyEditBox)
         {
           // задаем прозрачный фон
           SetBkMode ( ( HDC) wParam, TRANSPARENT);
           // цвет текста в списке, если необходимо
           SetTextColor ( ( HDC) wParam, RGB ( 200, 200, 27));
           // цвет фона текста тоже синий
           SetBkColor ( ( HDC) wParam, RGB ( 0, 128, 255));
          return ( LRESULT) hBrush;
         }
       }
       return true;
```

```
        case WM_DESTROY: // завершение работы программы
        {
          // удаляем логическую кисть
          DeleteObject ( hBrush);
        }
        return true;
     }
   return FALSE;
}
```

Рассмотрим еще одну возможность, поддерживаемую полем редактирования. Как вы знаете, данный элемент управления с успехом применяется для организации ввода пароля пользователя. Обычно реальные буквы и цифры пароля отображаются в виде символов звездочек '*', что затрудняет определение настоящего кода. В листинге 18.12 показан отрывок из программы, позволяющий просматривать скрытый пароль в поле редактирования.

Листинг 18.12. Просмотр скрытого пароля в поле редактирования

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
   // обработка сообщений
   switch ( message)
    {
     case WM_COMMAND:
           if ( HIWORD ( wParam) == EN_SETFOCUS)
           {
             if ( IsWindow ( ( HWND) lParam)) // если это окно
                  ShowPassword (); // пытаемся открыть пароль
           }
        break;
    }
   return FALSE;
}
// функция для просмотра пароля
void ShowPassword ()
{
   POINT point;
   // получаем текущие координаты курсора
   GetCursorPos ( &point);
   // определяем дескриптор окна
   HWND hX = WindowFromPoint ( point);
```

```
    // открываем пароль
    SendMessage ( hX, EM_SETPASSWORDCHAR, 0, 0);
    // обновляем поле редактирования
    RedrawWindow ( hX, NULL, NULL, RDW_ERASE | RDW_INVALIDATE);
}
```

Итак, мы перехватываем сообщение `EN_SETFOCUS` (фокус ввода для клавиатуры), посылаемое полю редактирования, и вызываем функцию `ShowPassword`. Хотя представленный пример не очень удобен в использовании, он полностью реализует поставленную задачу. Читателям же рекомендую применять "горячие" клавиши для вызова функции `ShowPassword`, когда курсор мыши находится в области окна с паролем.

18.1.4. Список

Стандартный список, как и раскрывающийся, помогает упорядочить большое количество отдельных записей, предоставляя удобный интерфейс для быстрого поиска и выбора нужного элемента.

Выбор шрифта ничем не отличается от рассмотренных ранее элементов управления, а работа с цветом показана в листинге 18.13.

Листинг 18.13. Использование цвета для стандартного списка

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
    // создаем дескриптор кисти
    static HBRUSH hBrush;
    // получаем дескриптор окна нашего списка
    HWND hMyListBox = GetDlgItem ( hwndDlg, IDC_MYLISTBOX);
    // обработка сообщений
    switch ( message)
     {
        case WM_INITDIALOG: // инициализация диалога
        {
          // создаем логическую кисть красного цвета
          hBrush = CreateSolidBrush ( RGB ( 255, 0, 0));
        }
        return true;
        // обработка цвета для списка
        case WM_CTLCOLORLISTBOX:
        {
          // выделяем сообщение для поля редактирования
          if ( ( HWND) lParam == hMyEditBox)
```

```
        {
          // задаем прозрачный фон
          SetBkMode ( ( HDC) wParam, TRANSPARENT);
          // цвет текста в списке, если необходимо
          SetTextColor ( ( HDC) wParam, RGB ( 0, 0, 0));
          // цвет фона текста тоже красный
          SetBkColor ( ( HDC) wParam, RGB ( 255, 0, 0));
         return ( LRESULT) hBrush;
        }
      }
      return true;
      case WM_DESTROY: // завершение работы программы
      {
        // удаляем логическую кисть
        DeleteObject ( hBrush);
      }
      return true;
   }
  return FALSE;
}
```

Далее мы рассмотрим довольно интересную возможность, позволяющую добавлять в обычный список значки (иконки). Для упрощения кода воспользуемся библиотекой MFC и напишем расширенный класс, производный от CListBox, но сначала изменим некоторые свойства нашего списка в редакторе ресурсов. Откройте вкладку **Styles** и выберите в списке **Owner draw** строчку **Fixed**, а также установите флажок **Has strings**. Это необходимо сделать, поскольку рисование и обработка текста в списке ложится полностью на нас. После этого создайте новый класс IconList и заполните его, как показано в листингах 18.14 и 18.15.

Листинг 18.14. Описание файла IconList.h

```
class IconList : public CListBox // производный от класса MFC
{
public:
    IconList (); // конструктор
    ~IconList () { } // пустой деструктор
// общедоступные функции
    // функция для определения иконок
    void AddIcons ( HICON hIcon);
    // функции для добавления новой записи в список
    void AddStringIcon ( LPCTSTR lpszText, unsigned int uIcon);
```

```
    void InsertStringIcon ( LPCTSTR lpszText, int iItem,
                            unsigned int uIcon);
    // добавим в класс обработку сообщения WM_DRAWITEM
    // { { AFX_VIRTUAL ( IconList)
    public:
    virtual void DrawItem ( LPDRAWITEMSTRUCT lpDrawItemStruct);
    // } } AFX_VIRTUAL
protected:
    // добавим класс для хранения иконок
    CImageList m_IconsList;
    // { { AFX_MSG ( IconList)
    // NOTE - the ClassWizard will add and remove member functions here.
    // } } AFX_MSG
    DECLARE_MESSAGE_MAP ()
}; // окончание класса
```

Листинг 18.15. Описание файла IconList.cpp

```
#include "stdafx.h"
#include "IconList.h"
// реализация класса
IconList :: IconList ()
{
   // создаем контейнер для хранения иконок
   m_IconsList.Create ( 16, 16, ILC_COLOR | ILC_MASK, 10, 10);
}
// пустая карта сообщений, созданная мастером
BEGIN_MESSAGE_MAP ( IconList, CListBox)
      // { { AFX_MSG_MAP ( IconList)
      // NOTE - the ClassWizard will add and remove mapping macros here.
      // } } AFX_MSG_MAP
END_MESSAGE_MAP ()
// реализация функций
void IconList :: AddIcons ( HICON hIcon)
{
   m_IconsList.Add ( hIcon);
}
void IconList :: AddStringIcon ( LPCTSTR lpszText, unsigned int uIcon)
{
   CListBox :: AddString ( lpszText);
}
void IconList :: InsertStringIcon ( LPCTSTR lpszText, int iItem,
                                    unsigned int uIcon)
```

```
{
   CListBox :: InsertString ( iItem, lpszText);
}
// функция рисования нашего списка
void IconList :: DrawItem ( LPDRAWITEMSTRUCT lpDrawItemStruct)
{
   CDC* pDC = CDC :: FromHandle ( lpDrawItemStruct->hDC);
   // определяем необходимые переменные
   CString buf; // буфер для текста
   COLORREF color; // цвет текста
   unsigned int uIcons;
   int iBgr = 0;
   CRect  Focus ( lpDrawItemStruct->rcItem);
   CRect  OutArea ( Focus);
   CRect  Text ( Focus);
   CPoint point ( Focus.left, Focus .top);
   // стиль для отображения текста
   unsigned int uStyle = DT_VCENTER | DT_LEFT;
   // устанавливаем режим фона
   iBgr = pDC->SetBkMode ( TRANSPARENT);
   // если элементов в списке нет, рисуем пустое поле
   if ( ( int) lpDrawItemStruct->itemID < 0)
   {
     if ( ( lpDrawItemStruct->itemAction & ODA_FOCUS) &&
                      ( lpDrawItemStruct->itemState & ODS_FOCUS))
     {
       // рисуем прямоугольник для фокуса
       pDC->DrawFocusRect ( &lpDrawItemStruct->rcItem);
     }
     else if ( ( lpDrawItemStruct->itemAction & ODA_FOCUS) &&
                      !( lpDrawItemStruct->itemState & ODS_FOCUS))
     {
       // рисуем прямоугольник для фокуса
       pDC->DrawFocusRect ( &lpDrawItemStruct->rcItem);
     }
     return;
   }
   // насчитываем отступ между текстом и иконкой
   Text.left += 20;
   Text.top += 2;
   OutArea.left += 20;
   // рисуем иконку в заданной области
    uIcons = ( unsigned int) lpDrawItemStruct->itemData;
```

```
  if ( m_IconsList)
      m_IconsList.Draw ( pDC, uIcons, point, ILD_NORMAL |
                              ILD_TRANSPARENT);
  // обрабатываем выделенные пункты списка
  if ( ( lpDrawItemStruct->itemState & ODS_SELECTED) &&
   ( lpDrawItemStruct->itemAction & ( ODA_DRAWENTIRE | ODA_SELECT)))
  {
    // используем системный цвет для заливки
    CBrush brush ( ::GetSysColor ( COLOR_HIGHLIGHT));
    pDC->FillRect ( &OutArea, &brush);
  }
  else if ( !( lpDrawItemStruct->itemState & ODS_SELECTED) &&
             ( lpDrawItemStruct->itemAction & ODA_SELECT))
  {
    CBrush brush ( ::GetSysColor ( COLOR_WINDOW));
    pDC->FillRect ( &OutArea, &brush);
  }
  // обрабатываем фокус для выбранного пункта
  if ( ( lpDrawItemStruct->itemAction & ODA_FOCUS) &&
      ( lpDrawItemStruct->itemState & ODS_FOCUS))
  {
    pDC->DrawFocusRect ( &Focus);
  }
  else if ( ( lpDrawItemStruct->itemAction & ODA_FOCUS) &&
           !( lpDrawItemStruct->itemState & ODS_FOCUS))
  {
    pDC->DrawFocusRect ( &Focus);
  }
  // обрабатываем цвет для текста
  if ( lpDrawItemStruct->itemState & ODS_SELECTED)
   color = pDC->SetTextColor ( ::GetSysColor ( COLOR_HIGHLIGHTTEXT));
  else if ( lpDrawItemStruct->itemState & ODS_DISABLED)
   color = pDC->SetTextColor ( ::GetSysColor ( COLOR_GRAYTEXT));
  else
   color = pDC->SetTextColor ( ::GetSysColor ( COLOR_WINDOWTEXT));
  // получаем текст и выводим его в список
  pDC->DrawText ( buf, -1, &Text, uStyle | DT_CALCRECT);
  pDC->DrawText ( buf, -1, &Text, uStyle);
  // восстанавливаем цвет текста и режим отображения
  pDC->SetBkMode ( iBgr);
  pDC->SetTextColor ( color);
}
```

Пример использования класса IconList показан в листинге 18.16. Замечу только, что желательно работать с иконками размером 16×16, поскольку для более крупных (32×32 или 48×48) придется пересчитывать размеры элементов в списке.

Листинг 18.16. Пример использования класса IconList

```
#include "IconList.h"
// создадим список производным от нашего класса IconList
IconList m_IconList;
// добавим необходимые иконки
m_IconList.AddIcons ( LoadIcon ( AfxGetInstanceHandle (),
                      MAKEINTRESOURCE(IDI_ICON1)));
m_IconList.AddIcons ( LoadIcon ( AfxGetInstanceHandle (),
                      MAKEINTRESOURCE(IDI_ICON2)));
m_IconList.AddIcons ( LoadIcon ( AfxGetInstanceHandle (),
                      MAKEINTRESOURCE(IDI_ICON3)));
// добавим в список значения
m_IconList.AddStringIcon ( "Audi", 0); // первая иконка
m_IconList.AddStringIcon ( "Ford", 1); // вторая иконка
m_IconList.AddStringIcon ( "Opel", 2); // третья иконка
m_IconList.InsertStringIcon ( "Audi", 3, 0); // первая иконка
```

Как видите, небольшая доработка стандартного списка позволила нам получить прекрасный графический интерфейс, намного улучшив восприятие информации.

И последнее, что мы рассмотрим, — это добавление возможности перетаскивания файлов в список посредством технологии "drag-and-drop". Для этого нам понадобится уже написанный ранее класс DragDrop. Единственное, что придется сделать, — это немного изменить функцию AddFile, как показано в листинге 18.17.

Листинг 18.17. Функция AddFile класса DragDrop для стандартного списка

```
// функция AddFile
void DragDrop :: AddFile ( HWND hControl, const char *File)
{
   // копируем в список только имена файлов
   char* s;
   s = strrchr ( File, '\\');
   char tmp[MAX_PATH];
   strcpy ( tmp, s+2);
```

```
   // добавляем имя файла в стандартный список
   SendMessage ( hControl, LB_ADDSTRING, 0, ( LPARAM) ( LPCTSTR) tmp);
   SendMessage ( hControl, LB_SETCURSEL, 0, 0);
}
```

Как вы заметили, все изменения коснулись только имен сообщений, используемых для стандартного списка: LB_ADDSTRING и LB_SETCURSEL.

Следующий элемент управления Rich Edit очень похож на поле редактирования (Edit Box), но имеет гораздо более широкие возможности. Например, на его основе можно легко создать полноценный текстовый процессор, не уступающий программам Word или WordPad. Здесь мы не будем рассматривать программирование данного элемента управления, поскольку для этого потребуется отдельная книга. Замечу только, что материал, рассмотренный ранее для Edit Box, практически полностью будет работать и с Rich Edit.

18.1.5. Линейка прокрутки

Этот элемент управления присутствует практически везде. Откройте любую папку или обозреватель Интернета, и вы обнаружите как минимум две линейки прокрутки: вертикальную и горизонтальную. Важность данного элемента трудно переоценить, ведь благодаря ему можно максимально быстро и удобно просматривать большие документы или графику. Даже производители манипулятора "мышь" добавили специальное колесико, управляющее прокруткой изображения в окне.

Чтобы изменить цвет фона линейки прокрутки, нужно обработать системное сообщение WM_CTLCOLORSCROLLBAR, как показано в листинге 18.18.

Листинг 18.18. Изменение цвета фона для линейки прокрутки

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
   // создаем дескриптор кисти
   static HBRUSH hBrush;
   // получаем дескриптор окна нашей линейки прокрутки
   HWND hMyScroll = GetDlgItem ( hwndDlg, IDC_MYSCROLL);
   // обработка сообщений
   switch ( message)
    {
      case WM_INITDIALOG: // инициализация диалога
      {
        // создаем логическую кисть оранжевого цвета
        hBrush = CreateSolidBrush ( RGB ( 255, 128, 0));
      }
```

```
        return true;
        case WM_CTLCOLORSCROLLBAR: // обработка цвета
        {
          // выделяем сообщение
          if ( ( HWND) lParam == hMyScroll)
          {
           // цвет фона оранжевый
           return ( LRESULT) hBrush;
          }
        }
        return true;
        case WM_DESTROY: // завершение работы программы
        {
          // удаляем логическую кисть
          DeleteObject ( hBrush);
        }
        return true;
     }
   return FALSE;
}
```

В большинстве случаев линейки прокрутки добавляются автоматически, и нет необходимости в их перепрограммировании. Замечу только, что для обработки сообщений от линейки следует использовать сообщения WM_HSCROLL и WM_VSCROLL.

18.1.6. Статический элемент

Данный элемент позволяет выводить на экран статический или динамический текст, отображать иконки, графические файлы, а также использоваться вместо кнопки. В большинстве случаев он применяется для оформления всевозможных надписей в окне программы. Чтобы статический элемент мог обрабатывать сообщения (например, нажатия мыши), в его свойствах следует установить флажок **Notify**.

Изменение отображаемого шрифта для этого элемента полностью аналогично описанному ранее для кнопки. Для изменения цвета можно применить код, как показано в листинге 18.19.

Листинг 18.19. Изменение цвета текста для статического элемента управления

```
// главная функция диалогового ( или обычного) окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
```

```
{
   // создаем дескриптор кисти
   static HBRUSH hBrush;
   // получаем дескриптор окна нашего элемента
   HWND hMyStatic = GetDlgItem ( hwndDlg, IDC_MYSTATIC);
   // обработка сообщений
   switch ( message)
    {
       case WM_INITDIALOG: // инициализация диалога
       {
         // создаем логическую кисть с системным цветом кнопки
         hBrush = CreateSolidBrush ( GetSysColor ( COLOR_BTNFACE));
       }
       return true;
       case WM_CTLCOLORSTATIC: // обработка цвета
       {
         // выделяем сообщение
         if ( ( HWND) lParam == hMyStatic)
         {
          // устанавливаем желаемый цвет текста и ( или) фона
          // цвет текста зеленый
          SetTextColor ( ( HDC) wParam, RGB ( 0, 128, 0));
          // цвет фона текста
          SetBkColor ( ( HDC) wParam, GetSysColor ( COLOR_BTNFACE));
          return ( LRESULT) hBrush;
         }
       }
       return true;
       case WM_DESTROY: // завершение работы программы
       {
         // удаляем логическую кисть
         DeleteObject ( hBrush);
       }
       return true;
    }
   return FALSE;
}
```

Несмотря на простоту, статический элемент управления может полностью преобразить внешний вид программы. Для примера создадим класс (производный от `CStatic`), позволяющий использовать различные трехмерные эффекты и нестандартные возможности. В листингах 18.20 и 18.21 представлены соответствующие файлы нового класса `MagicStatic`.

Листинг 18.20. Файл MagicStatic.h

```
// определение класса
class MagicStatic : public CStatic
{
public:
       MagicStatic ();
       ~MagicStatic ();
// общедоступные функции
       // установка нового текста
       void SetText ( const CString& Text);
       // установка рамки
       void SetBorder ( bool bOn);
       // преобразование в интернет-ссылку
       void SetLinkWWW ( bool bOn);
      // установка режима прозрачности
      void SetTrans ( bool bOn);
      // установка цвета текста
      void SetTextColor ( COLORREF color);
      // установка подтекстового фона
      void SetBgrTextColor ( COLORREF color);
      // установка трехмерного эффекта
      void Set3Deffect (bool bOn, int Effect);
      // установка разделителя
      void SetSeparator ( bool bOn);
       // виртуальные функции
       // { { AFX_VIRTUAL ( MagicStatic)
       // } } AFX_VIRTUAL
protected:
       void UpdateControl (); // функция перерисовки элемента
       HBRUSH m_Brush; // основная кисть
       bool m_bTrans; // признак прозрачности
       bool m_b3d; // признак трехмерности
       bool m_bSep; // признак разделителя
       int m_Effect; // номер эффекта
       COLORREF m_TextColor; // цвет текста по умолчанию
       COLORREF m_BgrTextColor; // цвет подтекста по умолчанию
       // карта сообщений класса
       // { { AFX_MSG ( MagicStatic)
       afx_msg void OnSysColorChange ();
       afx_msg void OnPaint ();
       afx_msg void OnLButtonDown ( UINT nFlags, CPoint point);
```

```
    // } } AFX_MSG
    DECLARE_MESSAGE_MAP ()
}; // окончание класса
```

Листинг 18.21. Файл MagicStatic.cpp

```
// реализация класса MagicStatic
#include "stdafx.h"
#include "MagicStatic.h"
// конструктор
MagicStatic :: MagicStatic ()
{
   m_Brush = ::CreateSolidBrush ( ::GetSysColor ( COLOR_3DFACE));
   m_TextColor = ::GetSysColor ( COLOR_WINDOWTEXT);
   m_BgrTextColor = RGB ( 0, 0, 0}; // белый цвет
   m_bSep = false;
   m_bTrans = false;
   m_b3d = false;
   m_Effect = 0;
}
// деструктор
MagicStatic :: ~MagicStatic ()
{
   // удаляем кисть
   ::DeleteObject ( m_Brush);
}
// карта сообщений
BEGIN_MESSAGE_MAP ( MagicStatic, Cstatic)
   // { { AFX_MSG_MAP ( MagicStatic)
   ON_WM_SYSCOLORCHANGE ()
   ON_WM_PAINT ()
   ON_WM_LBUTTONDOWN ()
   // } } AFX_MSG_MAP
END_MESSAGE_MAP ()
// функция обработки цвета
void MagicStatic :: OnSysColorChange ()
{
   Cstatic :: OnSysColorChange ();
   // удаляем старую кисть
   if ( m_Brush)
     ::DeleteObject ( m_Brush);
   // и создаем новую
   m_Brush = ::CreateSolidBrush ( GetSysColor ( COLOR_3DFACE));
```

```
   // перерисовываем наш элемент управления
   UpdateControl ();
}
// функция обработки нажатия левой кнопки мыши
void MagicStatic :: OnLButtonDown ( UINT nFlags, CPoint point)
{
   CString www;
   GetWindowText ( www);
   // вызываем системную функцию для открытия страницы
   ShellExecute ( NULL, "open", www, NULL, NULL, SW_SHOWNORMAL);
   Cstatic :: OnLButtonDown ( nFlags, point);
}
// функция рисования нашего статического элемента
void MagicStatic::OnPaint()
{
   // получаем контекст устройства рисования
   CPaintDC dc ( this);
   // объявляем необходимые переменные
   CString tmp;
   CRect rect;
   CBitmap bmp;
   CDC* pDC;
   DWORD dwPos = 0;
   UINT uMode;
   COLORREF colorText;
   // получаем размеры клиентской области окна
   GetClientRect ( rect);
   // сохраняем текст в переменную
   GetWindowText ( tmp);
   // если нужно отобразить разделитель, выполняем этот код
   if ( m_bSep)
   {
     CRect rcWin;
     // получаем полный размер элемента
     GetWindowRect ( rcWin);
     // задаем начальный режим отображения
     uMode = DT_TOP;
     // получаем стиль статического элемента
     DWORD dwStyle = GetStyle ();
     // рисуем прямоугольник
     dc.Draw3dRect ( 0, rcWin.Height () / 2, rcWin.Width (), 2,
                                ::GetSysColor ( COLOR_3DSHADOW),
                           ::GetSysColor ( COLOR_3DHIGHLIGHT));
```

```
    // обрабатываем текст в зависимости от текущего стиля
    if ( dwStyle & SS_CENTER)
    {
      tmp = "  " + tmp;
      tmp += "  ";
      uMode |= ( DT_CENTER | DT_VCENTER | DT_SINGLELINE);
    }
    else if ( dwStyle & SS_RIGHT)
    {
      tmp = "  " + tmp;
      uMode |= DT_RIGHT;
    }
    else
    {
      tmp += "  ";
      uMode |= DT_LEFT;
    }
    // сохраняем старый шрифт
    CFont* pOld = dc.SelectObject ( GetFont ());

     // устанавливаем системный цвет фона
     dc.SetBkColor ( ::GetSysColor ( COLOR_BTNFACE));
     // выводим текст
     dc.DrawText ( tmp, rect, uMode);
     // восстанавливаем старый шрифт
     dc.SelectObject ( pOld);
     // выходим из функции
     return;
}
// обрабатываем режим прозрачности
if ( !m_bTrans) // прозрачность используется
{
 // создаем новый совместимый контекст
 pDC = new CDC;
 pDC->CreateCompatibleDC ( &dc);
 bmp.CreateCompatibleBitmap ( &dc, rect.Width (), rect.Height ());
 pDC->SelectObject ( &bmp);
}
else // прозрачность не используется
{
 pDC = &dc;
}
// устанавливаем режим вывода
uMode = pDC->SetBkMode ( TRANSPARENT);
```

```
// устанавливаем требуемый цвет текста
colorText = pDC->SetTextColor ( m_TextColor);
// создаем сплошную кисть и заполняем прямоугольник
if ( !m_bTrans)
{
  CBrush br;
  br.Attach ( m_Brush);
  pDC->FillRect ( rect, &br);
  br.Detach ();
}
// выводим текст
dwPos = DT_CENTER;
pDC->DrawText ( tmp, rect, dwPos);
// обрабатываем выбранный эффект
if ( m_b3d)
{
  pDC->SetTextColor ( m_BgrTextColor);
  switch ( m_Effect)
  {
  case -4:
   rect.OffsetRect ( -4, -4);
   break;
  case -3:
   rect.OffsetRect ( -3, -3);
   break;
  case -2:
   rect.OffsetRect ( -2, -2);
   break;
  case -1:
   rect.OffsetRect ( -1, -1);
   break;
  case 0:
   rect.OffsetRect ( 0, 0);
   break;
  case 1:
   rect.OffsetRect ( 1, 1);
   break;
  case 2:
   rect.OffsetRect ( 2, 2);
   break;
  case 3:
   rect.OffsetRect ( 3, 3);
   break;
```

```
    case 4:
     rect.OffsetRect ( 4, 4);
     break;
  }
  // выводим текст
  pDC->DrawText ( tmp, rect, dwPos);
  }
  // восстанавливаем параметры контекста
  pDC->SetBkMode ( uMode);
  pDC->SetTextColor ( colorText);
  // если используется режим прозрачности, копируем данные
  if ( !m_bTrans)
  {
    dc.BitBlt ( 0, 0, rect.Width (), rect.Height (), pDC, 0,
                0, SRCCOPY);
    delete pDC;
  }
}
// функция обновления
void MagicStatic :: UpdateControl ()
{
  CRect ( rect);
  GetWindowRect ( rect);
  RedrawWindow ();
  GetParent ()->ScreenToClient ( rect);
  GetParent ()->InvalidateRect ( rect, true);
  GetParent ()->UpdateWindow ();
}
// общедоступные функции
void MagicStatic :: SetText ( const CString& Text)
{
  SetWindowText ( Text);
  UpdateControl ();
}
void MagicStatic :: SetTextColor ( COLORREF color)
{
  m_BgrTextColor = color;
  UpdateControl ();
}
void MagicStatic :: SetBorder ( bool bOn)
{
  if ( bOn)
    ModifyStyle ( 0, WS_BORDER, SWP_DRAWFRAME);
```

```
   else
     ModifyStyle ( WS_BORDER, 0, SWP_DRAWFRAME);
}
void MagicStatic :: SetBgrTextColor ( COLORREF color)
{
   m_TextColor = color;
   UpdateControl ();
}
void MagicStatic :: SetLinkWWW ( bool bOn)
{
   if ( bOn)
     ModifyStyle ( 0, SS_NOTIFY);
   else
     ModifyStyle ( SS_NOTIFY, 0);
}
void MagicStatic :: SetTrans ( bool bOn)
{
   m_bTrans = bOn;
   ModifyStyleEx ( 0, WS_EX_TRANSPARENT);
   UpdateControl ();
}
void MagicStatic :: Set3Deffect ( bool bOn, int Effect)
{
   m_b3d = bOn;
   m_Effect = Effect;
   UpdateControl ();
}
void MagicStatic :: SetSeparator( bool bOn)
{
   m_bSep = bOn;
   UpdateControl ();
}
```

Вот и получился несколько улучшенный класс для поддержки статического элемента управления. Несмотря на достаточно понятный интерфейс этого класса, приведу пример его использования, представленный в листинге 18.22.

Листинг 18.22. Пример работы с классом `MagicStatic`

```
// объявим переменную класса
MagicStatic m_MyStatic;
```

```
// красный текст и желтая тень
m_MyStatic.SetTextColor ( RGB ( 255, 0, 0));
m_MyStatic.SetBgrTextColor ( RGB ( 255, 255, 128));
// выпуклый текст
m_MyStatic.Set3Deffect ( true, 1);
// или создадим адресную ссылку
m_MyStatic.SetLinkWWW ( true);
```

Как видите, ничего сложного в работе с классом нет.

18.2. Дополнительные элементы управления

Кроме стандартных элементов управления, в Windows существует набор дополнительных, которые еще называют общими. К ним относятся: элемент анимации (Animate), управление датой и временем (Date Time Picker), расширенный раскрывающийся список (ComboBoxEx), счетчик (Spin Box), индикатор выполнения (Progress Bar), ползунок (Slider), управление горячими клавишами (Hot Key), список просмотра картинок и текста (List View), древовидный список (Tree View), управление вкладками (Tab), сетевой адрес (IP Address), окно свойств (Property Sheet), строка состояния (Status Bars), панель инструментов (Toolbar), окно подсказки (Tooltip), список изображений (Image List). Я не буду рассматривать все эти элементы управления подробно, поскольку об этом написано достаточно много книг. Здесь я расскажу о наиболее часто используемых элементах и приведу примеры, имеющие сугубо практический интерес для программистов.

Прежде всего необходимо инициализировать дополнительные элементы управления, чтобы их можно было без проблем применять в своей программе. Для этого предназначена специальная функция `InitCommonControlsEx`. Она имеет единственный аргумент — указатель на структуру `LPINITCOMMONCONTROLSEX`. Данная структура позволяет выбрать только те элементы, которые понадобятся в программе. В листинге 18.23 показано, как следует вызывать функцию инициализации.

Листинг 18.23. Инициализация дополнительных элементов управления

```
// главная функция программы
int APIENTRY WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPSTR lpCmdLine, int nCmdShow)
{
   MSG msg;
   INITCOMMONCONTROLSEX controls;
```

```
  // функция регистрации класса окна программы
  if ( !RegisterMyClassWindows ( hInstance))
    return ( FALSE);
  // заполняем поля структуры INITCOMMONCONTROLSEX
  controls.dwSize = sizeof ( INITCOMMONCONTROLSEX); // размер
  // выбираем элементы управления
  controls.dwICC = ICC_WIN95_CLASSES;
  // инициализируем системную библиотеку Comctl32.dll
  InitCommonControlsEx ( &controls);
  // выполняем другие необходимые действия
  // . . .
  // запускаем обработку сообщений Windows
  while ( GetMessage ( &msg, NULL, 0, 0))
  {
    TranslateMessage ( &msg);
    DispatchMessage ( &msg);
  }
  return ( msg.wParam);
}
```

Как видно из листинга, инициализацию общих элементов управления желательно выполнять после регистрации главного окна программы, в данном случае после функции `RegisterMyClassWindows`. Структура `INITCOMMONCONTROLSEX`, как я уже говорил, позволяет выбрать определенные элементы управления. В табл. 18.1 представлены возможные значения поля `dwICC`, которые могут комбинироваться между собой с помощью оператора ИЛИ.

Таблица 18.1. *Возможные значения поля* `dwICC`

Значение	Описание
ICC_WIN95_CLASSES	Animate, Hot Key, List View, Status Bar, Progress Bar, Tab, Tooltip, Toolbar, Tree View, Spin Box
ICC_USEREX_CLASSES	ComboBoxEx
ICC_UPDOWN_CLASS	Spin
ICC_TREEVIEW_CLASSES	Tree View
ICC_ANIMATE_CLASS	Animate
ICC_BAR_CLASSES	Status Bar, Toolbar, Tooltip
ICC_DATE_CLASSES	Date Time Picker
ICC_HOTKEY_CLASS	Hot Key
ICC_INTERNET_CLASSES	IP Address

Таблица 18.1 (окончание)

Значение	Описание
ICC_LISTVIEW_CLASSES	List View
ICC_PROGRESS_CLASS	Progress Bar
ICC_TAB_CLASSES	Tab

Если вы используете библиотеку MFC, вызывать функцию `InitCommonControlsEx` необязательно. А теперь приступим непосредственно к вопросам программирования дополнительных элементов управления.

18.2.1. Древовидный список

Данный элемент управления является, пожалуй, одним из самых используемых. В операционной системе Windows можно открыть любую папку, чтобы убедиться в этом. Все папки представлены в удобном для восприятия виде, и реализовано это с помощью древовидного списка (Tree View). Мы разберем наиболее интересный случай, связанный с отображением структуры всех логических дисков системы наподобие того, как это выглядит в папке **Мой компьютер**.

Прежде всего, нам понадобится создать вспомогательный класс, непосредственно работающий с древовидным списком. Если вы работаете с библиотекой MFC, можете просто взять за основу класс `CTreeView`. Для тех же, кто, как и я, предпочитает чистый Win32 API, приведу пример реализации такого класса (назовем его `CTree`) в листингах 18.24 и 18.25.

Листинг 18.24. Файл CTree.h

```
#include <commctrl.h>
// объявляем наш класс
class CTree
{
public:
    CTree ();
    ~CTree ();
// общедоступные функции
    // инициализируем дескриптор древовидного списка
    void InitHWNDTree ( HWND hTree);
    // получаем дескриптор древовидного списка
    HWND getHWND () { return m_hwnd; )
   // получаем дочерний элемент списка
   HTREEITEM GetChildItem ( HTREEITEM hItem) const;
```

```
// получаем номер выбранного изображения
bool GetItemImage ( HTREEITEM hItem, int& nImage,
                          int& nSelectedImage) const;
// получаем текст элемента списка
bool GetItemText ( HTREEITEM hItem, LPTSTR lpszText, UINT nBuffer)
                      const;
// получаем следующий элемент списка
HTREEITEM GetNextItem ( HTREEITEM hItem, UINT nCode) const;
// получаем следующий соседний элемент
HTREEITEM GetNextSiblingItem ( HTREEITEM hItem) const;
// получаем родительский элемент списка
HTREEITEM GetParentItem ( HTREEITEM hItem) const;
// получаем выделенный элемент
HTREEITEM GetSelectedItem () const;
// проверяем, является ли элемент дочерним
bool ItemHasChildren ( HTREEITEM hItem) const;
// подключить список изображений
HIMAGELIST SetImageList ( HIMAGELIST himl,
                          UINT nImage = TVSIL_NORMAL);
// установить указанный элемент списка
bool SetItem ( LPTVITEM pItem);
// второй вариант функции установки элемента списка
bool SetItem ( HTREEITEM hItem, UINT nMask, LPCTSTR lpszItem,
int nImage, int nSelectedImage, UINT nState, UINT nStateMask,
                                              LPARAM lParam);
 // установить изображение для указанного элемента списка
 bool SetItemImage ( HTREEITEM hItem, int nImage,int nSelectedImage);
 // установить состояние указанного элемента списка
 bool SetItemState ( HTREEITEM hItem, UINT nState, UINT nStateMask);
 // установить текст для указанного элемента списка
 bool SetItemText ( HTREEITEM hItem, LPCTSTR lpszItem);
 // удалить все элементы списка
 bool DeleteAllItems ();
// удалить указанный элемент списка
bool DeleteItem ( HTREEITEM hItem);
// развернуть древовидный список
bool Expand ( HTREEITEM hItem, UINT nCode = TVE_TOGGLE);
// вставить новый элемент в список
HTREEITEM InsertItem ( LPTVINSERTSTRUCT lpInsertStruct);
HTREEITEM InsertItem ( UINT nMask, LPCTSTR lpszItem, int nImage,
int nSelectedImage, UINT nState, UINT nStateMask, LPARAM lParam,
                       HTREEITEM hParent, HTREEITEM hInsertAfter);
HTREEITEM InsertItem ( LPCTSTR lpszItem, HTREEITEM hParent = TVI_ROOT,
                       HTREEITEM hInsertAfter = TVI_LAST);
```

```
   HTREEITEM  InsertItem ( LPCTSTR lpszItem, int nImage,
                     int nSelectedImage, HTREEITEM hParent = TVI_ROOT,
                                    HTREEITEM hInsertAfter = TVI_LAST);
   // выделить указанный элемент списка
   bool SelectItem ( HTREEITEM hItem);
   // загрузка файлов для выбранной папки
   bool LoadFiles ( HWND hList, char *Folder);
private:
   HWND m_hwnd; // дескриптор древовидного списка
}; // окончание класса
```

Листинг 18.25. Файл CTree.cpp

```
#include "stdafx.h"
#include "CTree.h"
// реализация класса CTree
CTree :: CTree ()
{
}
CTree :: ~CTree ()
{
   if ( m_hwnd)
     m_hwnd = NULL;
}
void CTree :: InitHWNDTree ( HWND hTree)
{
   m_hwnd = hTree;
}
HTREEITEM CTree :: GetChildItem ( HTREEITEM hItem) const
{
   return ( HTREEITEM) SendMessage ( m_hwnd, TVM_GETNEXTITEM,
            TVGN_CHILD, ( LPARAM) hItem);
}
bool CTree :: GetItemImage ( HTREEITEM hItem, int& nImage,
                              int& nSelectedImage) const
{
   TVITEM item;
   item.hItem = hItem;
   item.mask = TVIF_IMAGE | TVIF_SELECTEDIMAGE;
   bool bres = ( bool) SendMessage ( m_hwnd, TVM_GETITEM, 0,
                                     ( LPARAM) &item);
```

```
    if ( bres)
    {
      nImage = item.iImage;
      nSelectedImage = item.iSelectedImage;
    }
    return bres;
}
bool CTree :: GetItemText ( HTREEITEM hItem, LPTSTR lpszText,
                             UINT nBuffer) const
{
    TVITEM item;
    item.hItem = hItem;
    item.pszText = lpszText;
    item.cchTextMax = nBuffer;
    item.mask = TVIF_TEXT;
    return ( bool) SendMessage ( m_hwnd, TVM_GETITEM, 0,
            ( LPARAM) &item);
}
HTREEITEM CTree :: GetNextItem ( HTREEITEM hItem, UINT nCode) const
{
    return ( HTREEITEM) SendMessage ( m_hwnd, TVM_GETNEXTITEM, nCode,
                                      ( LPARAM) hItem);
}
HTREEITEM CTree :: GetNextSiblingItem ( HTREEITEM hItem) const
{
    return ( HTREEITEM) SendMessage ( m_hwnd, TVM_GETNEXTITEM, TVGN_NEXT,
                                      ( LPARAM) hItem);
}
HTREEITEM CTree :: GetParentItem ( HTREEITEM hItem) const
{
    return ( HTREEITEM) SendMessage ( m_hwnd, TVM_GETNEXTITEM,
                                        TVGN_PARENT, ( LPARAM) hItem);
}
HTREEITEM CTree :: GetSelectedItem () const
{
    return ( HTREEITEM) SendMessage ( m_hwnd, TVM_GETNEXTITEM,
                                        TVGN_CARET, 0);
}
bool CTree :: ItemHasChildren ( HTREEITEM hItem) const
{
    TVITEM item;
    item.hItem = hItem;
    item.mask = TVIF_CHILDREN;
```

```
   SendMessage ( m_hwnd, TVM_GETITEM, 0, ( LPARAM) &item);
   return item.cChildren;
}
HIMAGELIST CTree :: SetImageList ( HIMAGELIST himl, UINT nImage)
{
   if ( himl == 0) return 0;
   return ( HIMAGELIST) SendMessage ( m_hwnd, TVM_SETIMAGELIST, nImage,
                              ( LPARAM) ( UINT) ( HIMAGELIST) himl);
}
bool CTree :: SetItem ( LPTVITEM pItem)
{
   return ( bool) SendMessage ( m_hwnd, TVM_SETITEM, 0,
                                 ( LPARAM) pItem);
}
bool CTree :: SetItem ( HTREEITEM hItem, UINT nMask, LPCTSTR lpszItem,
         int nImage, int nSelectedImage, UINT nState, UINT nStateMask,
         LPARAM lParam)
{
   TVITEM ti;
   ti.mask = nMask;
   ti.hItem = hItem;
   ti.stateMask = nStateMask;
   ti.pszText = ( LPTSTR) lpszItem;
   ti.iImage = nImage;
   ti.state = nState;
   ti.iSelectedImage = nSelectedImage;
   ti.lParam = lParam;
   return ( bool) SendMessage ( m_hwnd, TVM_SETITEM, 0,
                                 ( LPARAM) &ti);
}
bool CTree :: SetItemImage ( HTREEITEM hItem, int nImage,
                              int nSelectedImage)
{
   return SetItem ( hItem, TVIF_IMAGE | TVIF_SELECTEDIMAGE, NULL, nImage,
                    nSelectedImage, 0, 0, NULL);
}
bool CTree :: SetItemState ( HTREEITEM hItem, UINT nState,
                              UINT nStateMask)
{
   return SetItem ( hItem, TVIF_STATE, NULL, 0, 0, nState, nStateMask,
                    NULL);
}
```

```
bool CTree :: SetItemText ( HTREEITEM hItem, LPCTSTR lpszItem)
{
   return SetItem ( hItem, TVIF_TEXT, lpszItem, 0, 0, 0, 0, NULL);
}
bool CTree :: DeleteAllItems ()
{
   return ( bool) SendMessage ( m_hwnd, TVM_DELETEITEM, 0,
                               ( LPARAM) TVI_ROOT);
}
bool CTree :: DeleteItem ( HTREEITEM hItem)
{
   return ( bool) SendMessage ( m_hwnd, TVM_DELETEITEM, 0,
                               ( LPARAM) hItem);
}
bool CTree :: Expand ( HTREEITEM hItem, UINT nCode)
{
   return ( bool) SendMessage ( m_hwnd, TVM_EXPAND, nCode,
                               ( LPARAM) hItem);
}
HTREEITEM CTree :: InsertItem ( LPTVINSERTSTRUCT lpInsertStruct)
{
   return ( HTREEITEM) SendMessage ( m_hwnd, TVM_INSERTITEM, 0,
                                    ( LPARAM) lpInsertStruct);
}
HTREEITEM CTree :: InsertItem ( UINT nMask, LPCTSTR lpszItem, int nImage,
         int nSelectedImage, UINT nState, UINT nStateMask, LPARAM lParam,
                              HTREEITEM hParent, HTREEITEM hInsertAfter)
{
   TVINSERTSTRUCT tis;
   tis.hParent = hParent;
   tis.item.iImage = nImage;
   tis.item.iSelectedImage = nSelectedImage;
   tis.item.state = nState;
   tis.item.stateMask = nStateMask;
   tis.hInsertAfter = hInsertAfter;
   tis.item.mask = nMask;
   tis.item.pszText = ( LPTSTR) lpszItem;
   tis.item.lParam = lParam;
   return ( HTREEITEM) SendMessage ( m_hwnd, TVM_INSERTITEM, 0,
                                    ( LPARAM) &tis);
}
HTREEITEM CTree :: InsertItem ( LPCTSTR lpszItem, HTREEITEM hParent,
                                HTREEITEM hInsertAfter)
```

```
{
   return InsertItem ( TVIF_TEXT, lpszItem, 0, 0, 0, 0, 0, hParent,
                       hInsertAfter);
}
HTREEITEM CTree :: InsertItem ( LPCTSTR lpszItem, int nImage,
         int nSelectedImage, HTREEITEM hParent, HTREEITEM hInsertAfter)
{
   return InsertItem ( TVIF_TEXT | TVIF_IMAGE | TVIF_SELECTEDIMAGE,
     lpszItem, nImage, nSelectedImage, 0, 0, 0, hParent, hInsertAfter);
}
bool CTree :: SelectItem ( HTREEITEM hItem)
{
   return ( bool) SendMessage ( m_hwnd, TVM_SELECTITEM, TVGN_CARET,
                               ( LPARAM) hItem);
}
bool CTree :: LoadFiles ( HWND hList, char* Folder)
{
   HANDLE hFind = NULL;
   WIN32_FIND_DATA fd;
   char *p = NULL;
   // получаем указатель на строку
   p = Folder;
   // обнуляем структуру поиска
   memset ( &fd, 0, sizeof ( WIN32_FIND_DATA));
try
{
   while ( *p) p++;
   // обрабатываем путь
   if ( strlen ( Folder) < 4) // если корневая папка
       strcpy ( p, "*.*"); // добавляем точку
   else // иначе
      strcpy ( p, "\\*.*"); // разделитель и точку
   // ищем первый файл или папку
   hFind = FindFirstFile ( Folder, &fd);
   // если файл или папка найдены
   if ( hFind != INVALID_HANDLE_VALUE)
   {
     do
     {
       // пропускаем ненужное
       if ( fd.cFileName[0] != '.' ||
           (fd.cFileName[1] != '.' && fd.cFileName[1]))
       {
         lstrcpy ( p + 1, fd.cFileName);
```

```
        // если найдена папка, переходим к дальнейшему поиску
        if ( fd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        {
          continue;
        }
        else
        {
          // если найден файл, добавляем его в список
          SendMessage ( hList, LB_ADDSTRING, 0,
                      ( LPARAM) ( LPCTSTR) fd.cFileName);
        }
      }
    // переходим к поиску следующего файла
    } while ( FindNextFile ( hFind, &fd));
   // закрываем дескриптор поиска
   FindClose ( hFind);
   }
   else
   {
     // не найден ни один файл
     FindClose ( hFind);
     return false;
   }
   p[0]=0;
}
catch ( . . .)
{
   // при любой ошибке выходим из функции
   if (hFind) FindClose ( hFind);
   return false;
}
   return true;
}
```

У нас получился базовый класс для управления древовидным списком. Теперь мы должны воспользоваться им для отображения структуры всех логических дисков, имеющихся на компьютере. Прежде всего, добавьте в редакторе ресурсов элемент управления **Tree Control**. На вкладке **Styles** установите следующие флажки: **Has buttons**, **Has lines**, **Lines at root**, **Show selection always**. Кроме этого, установите флажок **Scroll**, который находится на вкладке **More Styles**. Чтобы воспользоваться всеми преимуществами древовидного списка (Tree View), построим рабочий класс `CTreeWork`, производный от `CTree`. В листингах 18.26 и 18.27 представлены файлы данного клас-

са. Не забудьте добавить в опциях компоновщика ссылку на библиотеку Comctl32.lib.

Листинг 18.26. Файл CTreeWork.h

```
#include <commctrl.h>
#include <shlobj.h>
#include "CTree.h"
// объявление класса CTreeWork
class CTreeWork : public CTree
{
public:
   CEditorExt (); // конструктор
   ~CEditorExt(); // деструктор
// общедоступные функции
   // функция начальной инициализации
   void InitDrives ( HWND hDlg, bool bReset = true);
  // определяем все доступные логические диски
  void ExpandDrives ( HTREEITEM hParentItem, HWND hList);
  // перезагружаем древовидный список
  void DeleteAllChild ( HTREEITEM hItem,  bool IsEmpty = true);
  // определяем полный путь в списке
  void GetPathTree ( HTREEITEM hItem, char* Out, char* buffer);
private:
   HIMAGELIST m_im; // список изображений
   HTREEITEM m_Drv_Root; // корневой узел для дисков
   HTREEITEM m_Deskop_Root; // корневой узел для Рабочего стола
   // переменные для хранения строковых значений
   char m_Drv_Name[500];
   char m_Desktop_Path[350];
   // функции для добавления данных в древовидный список
   HTREEITEM AddItemTree ( const char *name, HTREEITEM hParentItem,
                           bool IsFolder, int Icon, int IconOpen);
   HTREEITEM AddItem( const char* path, HTREEITEM hParentItem,
                 bool IsFolder = false, int Icon = -1, int Icon2 = -1);
   // функция обновления изображений в списке
   void RefreshTree ( HTREEITEM hItem, int iImage);
}; // окончание класса
```

Листинг 18.27. Файл CTreeWork.cpp

```
#include "stdafx.h"
#incluse "CTreeWork.h"
```

```
// глобальная переменная дескриптора программы
HINSTANCE g_hInst;
// реализация класса CTreeWork
CTreeWork :: CTreeWork ()
{
// инициализируем переменные класса
   m_im = NULL;
   m_Drv_Root = NULL;
   m_Deskop_Root = NULL;
}
CTreeWork :: ~CTreeWork ()
{
   // удаляем список изображений из памяти
   ImageList_Destroy ( m_im);
   m_im = NULL;
}
HTREEITEM CTreeWork :: AddItemTree ( const char *name,
          HTREEITEM hParentItem, bool IsFolder, int Icon, int IconOpen)
{
   HTREEITEM hItem;
   // вставляем новое значение в список
   hItem = InsertItem ( name, Icon, IconOpen, hParentItem);
   // если корневая папка, текст опускаем
   if ( IsFolder) InsertItem ( "", hItem);
   return hItem;
}
HTREEITEM CTreeWork :: AddItem ( const char* path, HTREEITEM hParentItem,
                  bool IsFolder = false, int Icon = -1, int Icon2 = -1)
{
   // объявляем структуру оболочки для получения информации о папке
   SHFILEINFO shl_Info, shl_InfoOpen;
   // получаем информацию о маленькой иконке папки
   SHGetFileInfo ( path, NULL, &shl_Info, sizeof ( SHFILEINFO),
                  SHGFI_DISPLAYNAME | SHGFI_ICON | SHGFI_SMALLICON);
   // получаем информацию о маленькой иконке открытой папки
   SHGetFileInfo ( path, NULL, &shl_InfoOpen, sizeof ( SHFILEINFO),
    SHGFI_DISPLAYNAME |  SHGFI_ICON | SHGFI_SMALLICON | SHGFI_OPENICON);
   // выбираем нужную иконку и добавляем ее в список изображений
   int _Icon = Icon != -1 ? Icon : ImageList_AddIcon ( m_im,
                                                      shinfo.hIcon);
   int _IconOpen = Icon2 != -1 ? Icon2 : ImageList_AddIcon ( m_im,
                                                   shinfo_sel.hIcon);
```

```
   // определяем список изображений для древовидного элемента управления
   SetImageList ( m_im, LVSIL_NORMAL);
   return AddItemTree ( shinfo.szDisplayName, hParentItem, IsFolder,
                        Icon, Icon2);
}
void CTreeWork :: RefreshTree ( HTREEITEM hItem, int iImage)
{
   HTREEITEM hCurItem;
   int iNormal, iOpen;
   // определяем дочерний элемент списка
   hCurItem = GetChildItem ( hItem);
   // выполняем обработку
    while ( hCurItem)
    {
      // получаем картинки для двух состояний папки
      if ( GetItemImage ( hCurItem, iNormal, iOpen))
      {
        if ( iNormal > iImage) iNormal--;
        if ( iOpen > iImage) iOpen--;
        // устанавливаем значения картинок для элемента списка
        SetItemImage ( hCurItem, iNormal, iOpen);
      }
      // если есть еще дочерние элементы, обрабатываем и их
      if ( ItemHasChildren ( hCurItem) !=0)
      {
        // рекурсия
        RefreshTree ( hCurItem, iImage);
      }
      // переходим к следующему элементу
      hCurItem = GetNextSiblingItem ( hCurItem);
     } // end while
}
// реализация общих функций класса
void CTreeWork :: InitDrives ( HWND hDlg, bool bReset)
{
   LPITEMIDLIST item_List;
   char tmp[350];
   int iDesk_Icon, iMyComputer_Icon;
   // обновляем древовидный список
   if ( bReset)
     DeleteAllItems ();
   // очищаем список изображений
   SetImageList ( m_im, 0);
```

```
   // получаем путь к папке 'Рабочий Стол'
   SHGetSpecialFolderLocation ( hDlg, CSIDL_DESKTOP, &item_List);
   SHGetPathFromIDList ( item_List, tmp);
   // сохраняем путь в переменную класса
   strcpy ( m_Desktop_Path, tmp);
   // создаем новый список изображений
   m_im = ImageList_Create ( GetSystemMetrics ( SM_CXSMICON),
      GetSystemMetrics ( SM_CYSMICON), ILC_COLOR24 | ILC_MASK, 10, 1);
   // устанавливаем цвет фона для списка изображений
   ImageList_SetBkColor ( m_im, GetSysColor ( COLOR_WINDOW));
   // получаем корневые значки
   iDesk_Icon = ImageList_AddIcon ( m_im, ExtractIcon ( g_hInst,
                                    "shell32.dll", 34));
   iMyComputer_Icon = ImageList_AddIcon ( m_im, ExtractIcon( g_hInst,
                                          "shell32.dll", 15));
   // добавляем в древовидный список корневые значки
   m_Drv_Root = AddItem_Core ( "Drives", TVI_ROOT, true,
                              iMyComputer_Icon, iMyComputer_Icon);
   m_Deskop_Root = AddItem ( m_Desktop_Path, TVI_ROOT, true, iDesk_Icon,
                            iDesk_Icon);
}
void CTreeWork :: ExpandDrives ( HTREEITEM hParentItem, HWND hList)
{
   // если получен корневой узел
   if( hParentItem == m_Drv_Root)
   {
     char *pS;
     // удаляем дочерние элементы списка
     DeleteAllChild ( hParentItem, false);
     // получаем буквы всех установленных в системе дисков
     GetLogicalDriveStrings ( 500, m_Drv_Name);
     pS = m_Drv_Name;
     // заносим все найденные диски в древовидный список
     while ( *pS)
     {
       AddItem ( pS, m_Drv_Root, true);
       pS += strlen ( pS) + 1;
     }
   }
   else
   {
     // заносим в список имена найденных папок
     char path[500];
     char path2[MAX_PATH];
```

```
    char temp[MAX_PATH];
    WIN32_FIND_DATA ff;
    HANDLE hFind = NULL;
    // обнуляем структуру поиска
    memset ( &ff, 0, sizeof ( WIN32_FIND_DATA));
    // определяем путь к элементу списка
    GetPathTree ( hParentItem, path, "");
    // удаляем дочерние элементы списка
    DeleteAllChild ( hParentItem, false);
    // определяем путь для поиска папок
    strcpy ( path2, path);
    strcat ( path2, "*.*");
    // ищем папки
    if ( ( hFind = FindFirstFile ( path2, &ff)) ==
           INVALID_HANDLE_VALUE) return;
    do
    {
      if ( ff.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
      {
        // отсеиваем ненужное
        if ( ( strcmp(ff.cFileName, ".") != 0) &&
           ( strcmp ( ff.cFileName, "..") != 0))
        {
           // получаем имя найденной папки
           strcpy ( temp, path);
           strcat ( temp,  ff.cFileName);
           // создаем новый элемент списка для найденной папки
           HTREEITEM item = AddItem ( temp, hParentItem);
           InsertItem ( "", item);
           strcpy ( temp, "");
         }
      }
     // ищем следующую папку
     } while ( FindNextFile ( hFind, &ff));
     // завершаем поиск
     FindClose ( hFind);
     hFind = NULL;
   }
}
void CTreeWork :: DeleteAllChild ( HTREEITEM hItem,  bool IsEmpty)
{
   // получаем дочерний элемент списка
   HTREEITEM item_Child;
   item_Child = GetChildItem ( hItem);
```

```
   // обрабатываем полученный элемент
   while ( item_Child)
   {
     int icon1, icon2;
     // если дочерний элемент содержит свои дочерние элементы
     if ( GetChildItem ( item_Child))
     // выполняем рекурсию
          DeleteAllChild ( GetChildItem ( item_Child), isEmpty);
     // получаем оба значка для отображения элемента списка
     GetItemImage ( item_Child, icon1, icon2);
     // если необходимо, обновляем значки
     if ( icon2 != 0 && icon2 != icon1)
     {
       // удаляем значок из списка изображений
       ImageList_Remove ( m_im, icon2);
       // обновляем корневые узлы списка
       RefreshTree ( m_Drv_Root, icon2);
       RefreshTree ( m_Deskop_Root, icon2);
     }
     if ( icon1 != 0)
     {
       // удаляем значок из списка изображений
       ImageList_Remove ( m_im, icon1);
       // обновляем корневые узлы списка
       RefreshTreeImages (m_Drv_Root, icon1);
       RefreshTreeImages ( m_Deskop_Root, icon1);
     }
     // удаляем дочерний элемент из списка
     DeleteItem ( item_Child);
     // устанавливаем список изображений
     SetImageList ( m_im, TVSIL_NORMAL);
     // получаем следующий дочерний элемент
     item_Child = GetChildItem ( hItem);
   } // end while
   // если нужно, вставляем в список пустой элемент
   if ( isEmpty)
      InsertItem ( "", hItem);
}
void CTreeWork :: GetPathTree ( HTREEITEM hItem, char* Out,
                                char* buffer)
{
   char tmp[100];
   char tmp2[MAX_PATH];
```

```
// получаем родительский и дочерний элементы списка
HTREEITEM hParent = GetParentItem ( hItem);
HTREEITEM hChild = GetChildItem ( hItem);
// если это корневой узел, путь пустой
if ( hItem == m_Drv_Root)
{
  strcpy ( Out, "");
}
// если родительский элемент является корневым
if ( hParent == m_Drv_Root)
{
  HTREEITEM itmp = NULL;
  char *pS = m_Drv_Name;
  itmp = GetChildItem ( hParent);
  // получаем путь к элементу списка
  while ( itmp)
  {
    if ( itmp == hItem)
    {
      strcpy (tmp2, pS);
      strcat( tmp2, buffer);
      strcpy ( Out, tmp2);
      return; // выходим из функции
    }
    pS += strlen ( pS) + 1;
    // получаем следующий дочерний элемент
    itmp = GetNextItem ( itmp, TVGN_NEXT);
   }
}
// если элемент является корневым
else if ( hItem == m_Deskop_Root)
{
  char temp[MAX_PATH];
  // копируем путь к папке 'Рабочий стол'
  strcpy ( temp, m_Desktop_Path);
  strcat ( temp, "\\");
  if ( strcmp ( buffer, "") != 0)
  {
    strcat ( temp, buffer);
  }
  // сохраняем полученный путь и выходим
  strcpy ( Out, temp);
  return;
}
```

```
   else
   {
     // получаем текст элемента списка
     GetItemText ( hItem, tmp, sizeof ( tmp) + 1);
     if ( strcmp ( tmp, "") != 0)
     {
       strcat ( tmp, "\\");
       strcpy ( tmp2, tmp);
       strcat ( tmp2, buffer);
       strcpy ( Out, tmp2);
       // ищем слудующую часть пути
       GetPathTree ( hParent, Out, Out);
      }
   }
   // удаляем наклонную черту из пути
   if ( !hChild && strlen ( Out) > 0)
       strncpy ( Out, Out, strlen ( Out) - 1);
}
```

Класс `CTreeWork` может с успехом применяться для решения различных задач. Для примера приведу один вариант использования, который заключается в отображении всех файлов папки при выборе ее с помощью мыши. Чтобы упростить и уменьшить конечный код, вывод файлов будет осуществляться посредством стандартного списка (List Box), а не привычного для Windows списка изображений (List View). В листинге 18.28 показан отрывок программы, связанный с обработкой сообщений для диалогового окна.

Листинг 18.28. Использование класса `CTreeWork`

```
#include "stdafx.h"
#incluse "CTreeWork.h"
// класс CTreeWork
CTreeWork work;
// главная функция диалогового окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                          LPARAM lParam)
{
   char buffer[300];
   // получаем дескрипторы элементов управления
   HWND hTreeView = NULL, hListBox = NULL;
   hTreeView = GetDlgItem ( hwndDlg, IDC_MYTREEVIEW);
   hListBox = GetDlgItem ( hwndDlg, IDC_MYLISTBOX);
   // инициализируем класс CtreeWork для указанного древовидного списка
   work.InitHWNDTree ( hTreeView);
```

```
// обрабатываем сообщения
switch ( message)
{
   // обрабатываем сообщения для древовидного списка
   case WM_NOTIFY:
   {
      switch ( ( ( LPNMHDR) lParam)->code)
      {
        // одиночный или двойной щелчок мыши
        case NM_CLICK:
        case NM_DBLCLK:
        // проверяем элемент управления, на котором сделан щелчок
        if ( ( LOWORD ( wParam)) == IDC_MYTREEVIEW) // наш список
        {
          // получаем выбранный элемент списка
          HTREEITEM item = TreeView_GetSelection ( hTreeView);
          // определяем путь к этому элементу
          work.GetPathTree ( item, buffer, "");
          // очищаем стандартный список
          SendMessage ( hListBox, LB_RESETCONTENT, 0, 0);
          // выводим список файлов
          work.LoadFiles ( hListBox, buffer);
        }
        break;
      }
      // обрабатываем специфические сообщения для древовидного списка
      switch ( ( int) wParam)
      {
        case IDC_MYTREEVIEW: // выделяем наш список
        {
         NM_TREEVIEW *pNmTv = ( NM_TREEVIEW*) lParam;
         switch ( pNmTv->hdr.code)
         {
           case TVN_SELCHANGED: // выбран элемент списка
           {
             HTREEITEM item = TreeView_GetSelection ( hTreeView);
            work.GetPathTree ( item, buffer, "");
            SendMessage ( hListBox, LB_RESETCONTENT, 0, 0);
            work.LoadFiles ( hListBox, buffer);
           }
           break;
           case TVN_ITEMEXPANDED: // разворачивание списка
           {
             TV_ITEM itv = pNmTv->itemNew;
```

```
            // проверяем состояние
            if (itv.state & TVIS_EXPANDED)
               work.ExpandDrives ( itv.hItem, hListBox);
            else // удаляем элементы списка
               work.DeleteAllChild ( itv.hItem);
          }
          break;
        }
      }
    }
  }
  break;
  case WM_INITDIALOG:
        // начальная инициализация списка
        work.InitDrives ( hwndDlg);
  return true;
 }
 return false;
}
```

Как видите, обработка сообщений для древовидного списка не представляет собой ничего сложного. А теперь рассмотрим еще один не менее популярный элемент управления, предназначенный для отображения графической и текстовой информации List View.

18.2.2. Список просмотра картинок и текста

Несмотря на то, что работать со списком просмотра даже проще, чем с Tree View, в книгах по программированию почему-то о нем забывают. Я постараюсь восполнить этот пробел и расскажу подробнее о работе с этим интересным элементом. Прежде всего, следует заметить, что List View, как правило, используется совместно с Image List (список изображений). Вся информация в List View может быть представлена одним из четырех способов: крупные значки, мелкие значки, простой список и расширенный список. В первых трех случаях List View отображает только значки и названия, а в четвертом позволяет организовать для каждой записи несколько полей, отображающих различную текстовую и графическую информацию. Этот способ наиболее информативен и удобен для создания небольших баз данных и для управления файлами. Однако перейдем непосредственно к программированию.

Добавьте в редакторе ресурсов новый элемент **List Control** в окно своей программы. Настройте свойства элемента следующим образом: в окне **View** выберите из раскрывающегося списка строчку **Report**, установите флажки

Single selection и **Show selection always**. Вся остальная настройка будем выполнена нами программно. В листингах 18.29 и 18.30 показан пример класса `CFileListView`, использующий List View для работы с различными файлами.

Листинг 18.29. Файл FileListView.h

```
#include <commctrl.h>
// объявление класса CFileListView
class CFileListView
{
public:
   CFileListView (); // конструктор
   ~CFileListView (); // деструктор
// общие функции
   // установить дескриптор списка
   void SetHandleList ( HWND hListView);
   void InitViewList (); // инициализация
   // получаем номер выбранного элемента в списке
   int GetSelectedItem ();
   // возвращаем дескриптор используемого списка
   HWND GetViewList () { return m_hListView; }
   // сброс переменных
   void ResetMember ();
   // загрузка файла в список
   void OpenFile ();
private:
   // дескриптор элемента управления
   HWND m_hListView;
   // структуры для поддержки отображаемой в списке информации
   LV_COLUMN m_lvC; // описание столбца
   LV_ITEM m_lvI; // описание строки
   // дескриптор для списка изображений
   HIMAGELIST m_hSmall;
   int m_item; // счетчик элементов в списке
   __int64 m_fSize; // размер файла
// закрытые функции
   // загрузка иконки файла в список
   void LoadIconFile ( int index, int icon, char* Path);
   // вывод данных о файле в список
   void OutFileToList ( char* File);
   // определение размера файла в байтах
   __int64 GetFileSize64 ( char* File);
```

```
   // получение имени файла из пути
   void GetFileFromPath ( char *Path, char *File);
}; // окончание класса
```

Листинг 18.30. Файл FileListView.cpp

```
#include "stdafx.h"
#include <shellapi.h>
#include <Commdlg.h>
#include <io.h>
#include <fcntl.h>
#include <stdio.h>
#include "FileListView.h"
// реализация класса
CFileListView :: CFileListView ()
{
   m_hListView = NULL;
   m_item = 0;
   m_fSize = 0;
   m_hSmall = NULL;
}
CFileListView :: ~CFileListView ()
{
   // удаляем из памяти список изображений
   ImageList_Destroy ( m_hSmall);
   m_hSmall = NULL;
}
void CFileListView :: SetHandleList ( HWND hListView)
{
   if ( hListView) m_hListView = hListView;
}
void CFileListView :: InitViewList ()
{
   if ( !m_hListView) return;
   // обнуляем структуры
   memset ( &m_lvC, 0, sizeof ( LV_COLUMN));
   memset ( &m_lvI, 0, sizeof ( LV_ITEM));
   // формируем первый столбец
   m_lvC.mask = LVCF_TEXT | LVCF_SUBITEM | LVCF_WIDTH;
   m_lvC.cx = 200; // ширина в пикселах
   m_lvC.pszText = _T ( "File Name"); // название
   m_lvC.iSubItem = 0; // порядковый номер
```

```
   // вставляем первый столбец в список
   ListView_InsertColumn ( m_hListView, 0, &m_lvC);
   // формируем второй столбец
   m_lvC.cx = 150;
   m_lvC.pszText = _T ( "File Size");
   m_lvC.iSubItem = 1;
   // вставляем второй столбец в список
   ListView_InsertColumn ( m_hListView, 1, &m_lvC);
   // формируем третий столбец
   m_lvC.cx = 400;
   m_lvC.pszText = _T ( "File Path");
   m_lvC.iSubItem = 2;
   // вставляем третий столбец в список
   istView_InsertColumn ( m_hListView, 2, &m_lvC);
   // создаем список изображений ( иконок)
   m_hSmall = ImageList_Create ( GetSystemMetrics ( SM_CXSMICON),
        GetSystemMetrics ( SM_CYSMICON), ILC_COLOR | ILC_MASK, 1, 1);
}
int CFileListView :: GetSelectedItem ()
{
   if ( !m_hListView) return 0;
   return ListView_GetNextItem( m_hListView, -1, LVNI_ALL |
                              LVNI_SELECTED);
}
void CFileListView :: LoadIconFile ( int index, int icon, char* Path}
{
   if ( !m_hListView) return;
   SHFILEINFO shlInfo;
   HICON hIcon = NULL;
   ZeroMemory( &shlInfo, sizeof ( SHFILEINFO));
   // получаем значок для указанного файла
   SHGetFileInfo ( Path, 0, &shlInfo, sizeof ( SHFILEINFO), SHGFI_ICON
                 | SHGFI_SMALLICON);
   hIcon = shlInfo.hIcon;
   // добавляем значок в список изображений
   ImageList_AddIcon ( m_hSmall, hIcon);
   // устанавливаем список изображений для списка
   ListView_SetImageList ( m_hListView, m_hSmall, LVSIL_SMALL);
   // определяем свойства вставляемого значения
   m_lvI.mask = LVIF_TEXT | LVIF_IMAGE | LVIF_STATE;
   m_lvI.state = 0;
   m_lvI.stateMask = 0;
   m_lvI.iImage = icon;
```

```
    m_lvI.iItem = index;
    m_lvI.iSubItem = 0;
    // добавляем элемент в список
    ListView_InsertItem ( m_hListView, &m_lvI);
}
void CFileListView :: ResetMember ()
{
    ImageList_Remove ( m_hSmall, -1);
    m_item = 0;
    m_fSize = 0L;
    m_hSmall = NULL;
}
void CFileListView :: OpenFile ()
{
    char* szFilter = _T ( "All Files (*.*)\0*.*\0\0");
    char szPath [MAX_PATH] = "";
    OPENFILENAME of;
    // определяем тип открываемых файлов
    ZeroMemory ( &of, sizeof ( OPENFILENAME));
    of. lStructSize = sizeof ( OPENFILENAME);
    of. hwndOwner = m_hListView;
    of. lpstrFile = szPath;
    of. nMaxFile = sizeof ( szPath);
    of. lpstrFilter = szFilter;
    of. Flags = OFN_HIDEREADONLY | OFN_FILEMUSTEXIST;
    // вызываем диалог для открытия файла
    if ( GetOpenFileName ( &of))
    {
        OutFileToList ( of. lpstrFile);
    }
}
void CFileListView :: OutFileToList ( char* File)
{
    if ( !m_hListView) return;
    char szFileName[100];
    char szSize[30];
    // загружаем значок файла в список
    LoadIconFile ( m_item, m_item, File);
    // выделяем имя файла из пути
    GetFileFromPath ( File, szFileName);
    // записываем имя файла в первый столбец списка
    ListView_SetItemText ( m_hListView, m_item, 0, szFileName);
    // определяем размер файла в байтах
    m_fSize = GetFileSize64 ( File);
```

```
    // форматируем значение размера файла
    sprintf ( szSize, _T ( "%i"), m_fSize);
    // и записываем во второй столбец списка
    ListView_SetItemText ( m_hListView, m_item, 1, szSize);
    // записываем путь к файлу в третий столбец списка
    ListView_SetItemText ( m_hListView, m_item, 2, File);
    // увеличиваем счетчик файлов в списке
    m_item ++;
    m_fSize = 0L;
}
__int64 CFileListView :: GetFileSize64 ( char* File)
{
    int fl = 0;
    __int64 i64 = 0;
    // открываем файл
    if ( ( fl  = _open ( File, _O_RDONLY | _O_BINARY)) != -1)
    {
      // определяем размер файла
      i64 = _filelengthi64 ( fl);
      _close ( fl);
      return i64;
    }
    return 0L;
}
void CFileListView :: GetFileFromPath ( char *Path, char *File)
{
    // получаем указатель на строковое значение пути
    char *pS = Path + strlen ( Path);
    // обрабатываем строковое значение для пути
     while ( pS > Path)
     {
       if ( *pS == ':' || *pS == '\\')
       {
         pS++;
         break;
       }
       pS--;
     }
    // сохраняем имя файла
    strcpy ( File, pS);
}
```

Теперь рассмотрим, как можно применить класс `CFileListView` в своей программе. Как и прежде, мы будем использовать диалоговую функцию в каче-

стве основной функции окна. В листинге 18.31 показан отрывок из программы, в котором открытые файлы загружаются в список List View.

Листинг 18.31. Пример использования класса `CFileListView`

```
#include "stdafx.h"
#include "FileListView.h"
// класс CFileListView
CFileListView flv;
// главная функция диалогового окна
BOOL CALLBACK MyDlgProc ( HWND hwndDlg, UINT message, WPARAM wParam,
                           LPARAM lParam)
{
   // получаем дескриптор списка
   HWND hListView = NULL;
   hListView = GetDlgItem ( hwndDlg, IDC_MYLISTVIEW);
   // обрабатываем сообщения
   switch ( message)
   {
   case WM_INITDIALOG:
            // указываем дескриптор списка
            flv.SetHandleList ( hListView);
            // инициализируем список
            flv.InitViewList ();
      return true;
   }
   case WM_COMMAND:
       // если нажата кнопка открытия файла, обрабатываем сообщение
       if ( LOWORD ( wParam) == IDC_OPENFILE)
       {
         // открываем файл и заносим в список
         flv.OpenFile ();
         return true;
       }
   break;
   return false;
}
```

В рассмотренном примере предполагается наличие кнопки для открытия и загрузки файла в список. Сначала мы записываем в наш класс дескриптор элемента управления List View, а затем инициализируем список. Далее, пользуясь кнопкой открытия файла (например, `IDC_OPENFILE`), загружаем любой выбранный файл в список. Каждая запись состоит из трех частей:

имя файла, размер файла и полный путь к файлу. При необходимости вы можете самостоятельно добавить различные функциональные возможности.

18.2.3. Окно свойств

Окно свойств (Property Sheet) является идеальным элементом управления для создания как инсталляционных приложений, так и обычных программ, работающих по принципу пошагового выполнения (мастера). Мы рассмотрим первый вариант использования данного элемента и напишем простую программу установки, но прежде я хотел бы объяснить главный принцип организации окна свойств. Он состоит в том, что несколько отдельных независимых диалоговых окон размещаются на общем родительском окне, но одновременно доступен только один из них. Каждое дочернее окно имеет свою оконную функцию, в котором обрабатываются как собственные сообщения, так и сообщения родительского окна. На этом построено взаимодействие между родительским и дочерними окнами.

Прежде всего, нам понадобится создать основной каркас программы, который будет играть роль фонового. В листинге 18.32 показано, как это нужно сделать.

Листинг 18.32. Создание каркаса для программы установки

```
#include "stdafx.h"
#include <shlobj.h>
#include <shfolder.h>
#include <prsht.h>
#include <Wingdi.h>
// глобальные переменные
HINSTANCE hInst = NULL;
// набор цветов для градиентной заливки
enum
{
nRed = 164, nGreen = 45, nBlue = 189,
nRed2 = 225, nGreen2 = 174, nBlue2 = 236
};
// глобальные функции
BOOL InitApplication ( HINSTANCE hInst);
BOOL InitWindowInstall ( HINSTANCE hInst, int iShow);
LONG APIENTRY MainWndProc ( HWND, UINT, UINT, LONG);
void FadeVertical ( HWND hWnd, COLORREF Clr1, COLORREF Clr2, int nX1,
                    int nY1, int nX2, int nY2);
void OutTextScreen ( HWND hWnd, HDC hDC, RECT Rect);
```

```
// функция WinMain
int APIENTRY WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                       LPSTR lpCmdLine, int nCmdShow)
{
   MSG msg;
   hInst = hInstance; // копируем дескриптор программы в переменную
   // инициализируем окно программы
   if ( !InitApplication ( hInstance))
         return ( FALSE);
   // инициализируем дополнительные элементы управления
   InitCommonControls ();
   // создаем основное окно программы
   if ( !InitWindowInstall ( hInstance, nCmdShow))
          return ( FALSE);
   // запускаем цикл обработки сообщений
   while ( GetMessage ( &msg, NULL, 0, 0))
   {
     TranslateMessage ( &msg);
     DispatchMessage ( &msg);
   }
   return ( msg.wParam);
}
BOOL InitApplication ( HINSTANCE hInstance)
{
   WNDCLASS wc;
   memset ( &wc, 0, sizeof ( WNDCLASS));
   // определяем параметры основного окна программы
   wc.style = CS_VREDRAW | CS_HREDRAW;
   // определяем оконную функцию
   wc.lpfnWndProc = ( WNDPROC) MainWndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = NULL;
   wc.hCursor = NULL;
   wc.hbrBackground = ( HBRUSH) COLOR_BACKGROUND;
   wc.lpszMenuName =  NULL;
   // указываем уникальное имя класса окна
   wc.lpszClassName = TEXT ( "MyInstallClass");
   // регистрируем наш класс окна в системе
   return ( RegisterClass ( &wc));
}
```

```
BOOL InitWindowInstall ( HINSTANCE hInstance, int nCmdShow)
{
   HWND hWndMain;
   // создаем окно на весь экран
   hWndMain = CreateWindowEx ( WS_EX_CONTROLPARENT,
      TEXT ( "MyInstallClass"), TEXT ( "Install My Program"),
      WS_POPUPWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 9999, 9999,
      GetDesktopWindow (), NULL, hInstance, NULL);
   // если окно не создано, возвращаем ошибку
   if ( !hWndMain)
       return ( FALSE);
   // выводим окно на экран дисплея
   ShowWindow ( hWndMain, nCmdShow);
   // обновляем окно
   UpdateWindow ( hWndMain);
   return ( TRUE);
}
// главная оконная функция программы
LONG APIENTRY MainWndProc ( HWND hWnd, UINT message, UINT wParam,
                            LONG lParam)
{
   PAINTSTRUCT ps;
   HDC  hdc = NULL;
   RECT Rect;
   // обрабатываем сообщения
   switch ( message)
   {
    case WM_CREATE:
         // отображаем окно в развернутом виде
         ShowWindow ( hWnd, SW_SHOWMAXIMIZED);
         return 0;
    // можно оставить обработку системного меню
    case WM_SYSCOMMAND :
         switch ( wParam & 0xFFF0)
         {
          case SC_CLOSE:
             PostQuitMessage ( 0);
             break;

          default:
            break;
         }
         break;
```

```
   // перерисовываем окно с градиентной заливкой
   case WM_PAINT:
      // начинаем рисование
      hdc = BeginPaint ( hWnd, &ps);
      // получаем размеры клиентской области окна
      GetClientRect ( hWnd, &rRect);
      // рисуем градиентную заливку
      FadeVertical ( hWnd, RGB ( nRed, nGreen, nBlue),
     RGB ( nRed2, nGreen2, nBlue2), ( int) Rect.top, ( int) Rect.left,
    ( int) Rect.right, ( int) Rect.bottom);
      // проверяем область вывода и освобождаем контекст
      ValidateRect ( hWnd, NULL);
      EndPaint ( hWnd, &ps);
   // завершение работы программы
   case WM_DESTROY:
        PostQuitMessage ( 0);
        break;
   // все остальные сообщения обрабатываются здесь
   default:
     return ( DefWindowProc ( hWnd, message, wParam, lParam));
   }
   return ( 0);
}
// функция градиентной заливки по вертикали
void FadeVertical ( HWND hWnd, COLORREF Clr1, COLORREF Clr2, int nX1,
                    int nY1, int nX2, int nY2)
{
   HBRUSH hBrush, hBrushTmp;
   HPEN hPen, hPenTmp;
   HDC hDC = NULL;
   COLORREF clrTmp1, clrTmp2;
   RECT Rect;
   int i, iTotal;
   double red1,red2,redCurrent, green1, green2, greenCurrent, blue1,
          blue2, blueCurrent, red, green, blue;
   // получаем компоненты цвета
   red1 = GetRValue ( Color1);
   green1 = GetGValue ( Color1);
   blue1 = GetBValue ( Color1);
   red2 = GetRValue ( Color2);
   green2 = GetGValue ( Color2);
   blue2 = GetBValue ( Color2);
   // вычисляем размер по вертикали
   iTotal = nY2 - nY1 + 1;
```

```
// корректируем среднее значение цветов
red = ( red2 - red1) / ( iTotal - 1);
green = ( green2 - green1) / ( iTotal - 1);
blue = ( blue2 - blue1) / ( iTotal - 1);
// сохраняем текущие значения цвета
redCurrent = red1;
greenCurrent = green1;
blueCurrent = blue1;
// получаем контекст устройства и размеры области рисования
hDC = GetDC ( hWnd);
GetClientRect ( hWnd, &Rect);
// записываем текущие значения цвета
clrTmp1 = RGB ( ( int) redCurrent, ( int) greenCurrent,
               ( int) blueCurrent);
clrTmp2 = RGB ( ( int) redCurrent, ( int) greenCurrent,
               ( int) blueCurrent);
// рисуем заливку
for ( i = nY1; i <= nY2; i++)
{
  if ( ( i <= Rect.bottom) && ( i >= 0))
  {
    // создаем логические перо и кисть
    hPen = CreatePen ( PS_SOLID, 0, clrTmp1);
    hBrush = CreateSolidBrush ( clrTmp2);
    // устанавливаем их для рисования
    hPenTmp = ( HPEN) SelectObject ( hDC, hPen);
    hBrushTmp = ( HBRUSH) SelectObject ( hDC, hBrush);
    // рисуем прямоугольник
    Rectangle ( hDC, nX1, i, nX2, i + 1);
    // освобождаем ресурсы
    DeleteObject ( hPenTmp);
    DeleteObject ( hBrushTmp);
    // сохраняем новые значения цветов
    redCurrent += red;
    greenCurrent += green;
    blueCurrent += blue;
  }
}
// устанавливаем прозрачный фон
SetBkMode ( hDC, TRANSPARENT);
// выводим на экран текст
OutTextScreen ( hWnd, hDC, Rect);
// восстанавливаем параметры контекста и освобождаем ресурсы
ReleaseDC ( hWnd, hDC);
```

```
    DeleteObject ( hPen);
    DeleteObject ( hBrush);
    // перерисовываем окно
    InvalidateRect ( hWnd, 0, false);
}
// функция для отображения текста на экране
void OutTextScreen ( HWND hWnd, HDC hDC, RECT Rect)
{
    // имя программы
    LPCTSTR lpszNameProg = TEXT ( "My Programm  v.1.0");
    // авторские права на программу
    LPCTSTR lpszCopy = TEXT ( "Copyright(C) 2004 My Company, Ltd.");
    static HFONT  hFont;
    static LOGFONT lf;
    int y = 0;
    // обнуляем структуру LOGFONT
    ZeroMemory ( &lf, sizeof ( LOGFONT));
    // определяем размер экрана по вертикали
    y = GetSystemMetrics ( SM_CYSCREEN);
    // определяем параметры шрифта
    lf.lfWeight = FW_BOLD;
    lf.lfHeight = 48;
    lstrcpy ( lf.lfFaceName, TEXT ( "Arial"));
    // создаем шрифт
    hFont = CreateFontIndirect ( &lf);
    // выбираем созданный шрифт, выводим на экран теневой текст
    SelectObject ( hDC, hFont);
    TextOut ( hDC, Rect.top + 30, Rect.left + 30, lpszName,
              lstrlen ( lpszName));
    // выводим на экран текст переднего плана
    hFont = CreateFontIndirect ( &lf);
    SelectObject ( hDC, hFont);
    SetTextColor ( hDC, RGB ( 255, 255, 0));
    TextOut ( hDC, Rect.top + 32, Rect.left + 32, lpszName,
              lstrlen ( lpszName));
    // выводим на экран авторские права
    lf.lfHeight = 16;
    hFont = CreateFontIndirect ( &lf);
    SelectObject ( hDC, hFont);
    TextOut ( hDC, Rect.top + 30, yScreen - 30, lpszCopy,
              lstrlen ( lpszCopy));
    // удаляем ресурс шрифта
    DeleteObject ( hFont);
}
```

Теперь у нас получился каркас программы, на фоне которого будет работать программа установки. Для приближения к профессиональным пакетам инсталляции была добавлена возможность градиентной заливки и вывод информации на экран. Следующим шагом будет создание диалоговых окон и окна свойств. В редакторе ресурсов создайте необходимое число одинаковых по размеру диалоговых окон (для примера будет использовано три диалога). В свойствах каждого диалога установите следующие опции: выберите из списка стиль **Child** и тип рамки **Thin**, установите флажки **Title Bar** и **Disabled**. Добавьте в исходный код (предполагается, что существует один общий файл программы) оконные функции для диалогов и функцию создания окна свойств, как показано в листинге 18.33. Сразу замечу, что в коде представлены только вносимые изменения, а остальная часть пропущена (см. листинг 18.32). Первый диалог будет служить для приглашения к установке программы. Второй позволит пользователю ознакомиться с лицензией и сделать выбор, а третий диалог начнет непосредственно инсталляцию программы на компьютер пользователя. Самостоятельно вы можете добавить промежуточные диалоговые окна (например, для выбора папки). Первый и третий диалоги оформите по своему усмотрению, а на второй добавьте поле редактирования (установите флажки **Multiline**, **Vertical scroll**, **Horizontal scroll**, **Want return** и **Read-only**) и два переключателя (Radio Button) для выбора пользователя (согласен или не согласен). Кроме этого, напишите текст лицензии и сохраните его в текстовый файл. С помощью редактора ресурсов добавьте этот файл в ресурсы своей программы.

Листинг 18.33. Создание окна свойств

```
// объявляем диалоговые функции
BOOL APIENTRY WelcomeDlg ( HWND, UINT, UINT, LONG); // приветствие
BOOL APIENTRY LicenseDlg ( HWND, UINT, UINT, LONG); // лицензия
BOOL APIENTRY InstallDlg ( HWND, UINT, UINT, LONG); // установка
// функция создания окна свойств
int CreateWizard ( HWND hWnd, HINSTANCE hInst);
// функция настройки страниц свойств
void FillPage( PROPSHEETPAGE*, int, LPSTR, DLGPROC);
// функция обработки событий для окна свойств
int CALLBACK PropProc ( HWND propDlg, UINT uMsg, LPARAM lParam);
// функция обработки нажатия кнопки Cancel
bool ClickCancel ( HWND hWnd);
// функция загрузки текста лицензионного соглашения
int LoadLicense ( HWND hEdit);
// добавляем функцию CreateWizard в общий код программы
LONG APIENTRY MainWndProc ( HWND hWnd, UINT message, UINT wParam,
                            LONG lParam)
```

```
{
// . . .
// обрабатываем сообщения
   switch ( message)
   {
    case WM_CREATE:
         // отображаем окно в развернутом виде
         ShowWindow ( hWnd, SW_SHOWMAXIMIZED);
         // рисуем окно свойств
         CreateWizard ( hWnd, hInst);
         return 0;
   // . . .
   // все остальные сообщения обрабатываются здесь
   default:
     return ( DefWindowProc ( hWnd, message, wParam, lParam));
   }
   return ( 0);
}
// функция создания окна свойств
int CreateWizard ( HWND hWnd, HINSTANCE hInst)
{
   // определяем структуры окна свойств
   PROPSHEETPAGE psp[4]; // всегда на 1 больше числа диалогов
   PROPSHEETHEADER psh;
   // заполняем страницы свойств
   FillInPropertyPage ( &psp[0], IDD_WELCOME, TEXT ( "Welcome"),
                        WelcomeDlg);
   FillInPropertyPage ( &psp[1], IDD_LICENSE, TEXT ( "License"),
                        LicenseDlg);
   FillInPropertyPage ( &psp[2], IDD_INSTALL, TEXT ( "Ready Install"),
                        InstallDlg);
   // заполняем заголовок для окна свойств
   psh.dwSize = sizeof ( PROPSHEETHEADER);
   psh.dwFlags = PSH_PROPSHEETPAGE | PSH_WIZARD | PSH_NOAPPLYNOW |
                PSH_USECALLBACK | PSH_USEHICON;
   psh.hwndParent = hWnd;
   // заголовок окна свойств
   psh.pszCaption = ( LPSTR) TEXT ( "Install My Programm");
   psh.nPages = sizeof ( psp) / sizeof ( PROPSHEETPAGE);
   // функция обработки событий для окна свойств
   psh.pfnCallback = PropProc;
   // загружаем иконку, предварительно сохраненную в файле ресурсов
   psh.hIcon = LoadIcon ( hInst, MAKEINTRESOURCE ( IDI_MYICON));
   psh.ppsp = ( LPCPROPSHEETPAGE) &psp;
```

```
   // создаем окно свойств
   return ( PropertySheet ( &psh));
}
void FillPage( PROPSHEETPAGE* ps, int iDlg_ID, LPSTR lpTitle,
               DLGPROC pfnDlg)
{
   ps->dwSize = sizeof ( PROPSHEETPAGE);
   ps->dwFlags = 0;
   ps->hInstance = hInst;
   // идентификатор диалога, созданного редактором ресурсов
   ps->pszTemplate = MAKEINTRESOURCE ( idDlg_ID);
   ps->pszIcon = NULL;
   // указатель на диалоговую функцию
   ps->pfnDlgProc = pfnDlg;
   // заголовок страницы
   ps->pszTitle = lpTitle;
   ps->lParam = 0;
}
// функция обработки событий для окна свойств
int CALLBACK PropProc( HWND propDlg, UINT uMsg, LPARAM lParam)
{
   // если нужно, обрабатываем сообщения окна свойств
   switch ( uMsg)
   {
    case PSCB_PRECREATE: // окно свойств должно быть создано
         break;
   case PSCB_INITIALIZED: // инициализация окна свойств
         break;
   }
  return TRUE;
}
// первое окно программы установки
BOOL APIENTRY WelcomeDlg (HWND hDlg, UINT message, UINT wParam,
                          LONG lParam)
{
   // обработка сообщений
   switch ( message}
   {
    case WM_SHOWWINDOW:
        // центруем окно свойств на экране
        DialogCenter ( hDlg);
        break;
```

```
   case WM_NOTIFY:
   {
     LPNMHDR lpnm = ( LPNMHDR) lParam;
     // обрабатываем сообщения страницы свойств
     switch ( lpnm->code)
     {
      case PSN_KILLACTIVE: // страница закрывается
         // устанавливаем возвращаемое значение
         SetWindowLong ( hDlg, DWL_MSGRESULT, FALSE);
         return TRUE;
      case PSN_SETACTIVE: // инициализация страницы свойств
         // делаем активной кнопку Next
         PropSheet_SetWizButtons ( GetParent ( hDlg), PSWIZB_NEXT);
         break;
      case PSN_WIZNEXT: // нажата кнопка Next
         break;
      case PSN_WIZBACK: // нажата кнопка Back
         break;
      case PSN_RESET: // окно свойств будет закрыто
         break;
      case PSN_APPLY: // нажата одна из кнопок: OK, Apply, Close
         break;
      case PSN_QUERYCANCEL: // нажата кнопка закрытия окна свойств
         if ( ClickCancel ( hDlg) == false) // нажата кнопка отмены
         {
           // устанавливаем возвращаемое значение
           SetWindowLong ( hDlg, DWL_MSGRESULT, TRUE);
         }
         else // завершаем программу
         {
           PostQuitMessage ( 0);
           return FALSE;
         }
         return TRUE;
     }
   }
     break;
   default:
     return FALSE;
  }
  return TRUE;
}
```

```
// функция обработки нажатия кнопки Cancel
bool ClickCancel ( HWND hWnd)
{
   if ( MessageBox ( hWnd, "Для отмены установки нажмите кнопку Отмена.",
     TEXT ( "Установка"), MB_ICONQUESTION | MB_OKCANCEL | MB_DEFBUTTON)
                           == IDCANCEL)
   {
     return true; // завершить установку программы
   }
   return false; // продолжить установку программы
}
// второе окно программы установки
BOOL APIENTRY LicenseDlg ( HWND hDlg, UINT message, UINT wParam,
                           LONG lParam)
{
   HWND hEdit = NULL;
   // обработка сообщений
   switch ( message)
   {
   case WM_NOTIFY:
    {
      LPNMHDR lpnm = ( LPNMHDR) lParam;
      // обрабатываем сообщения страницы свойств
      switch ( lpnm->code)
      {
       case PSN_SETACTIVE: // инициализация страницы свойств
          // делаем активной кнопку Back
          PropSheet_SetWizButtons ( GetParent ( hDlg), PSWIZB_BACK);
          // получаем дескриптор окна ввода
          hEdit = ( GetDlgItem ( hDlg, IDC_MYEDIT));
          // загружаем текст лицензионного соглашения
          LoadLicense ( hEdit);
          // устанавливаем начальное состояние радиокнопок
          CheckRadioButton ( hDlg, IDC_YES, IDC_NO, IDC_NO);
          break;
       case PSN_QUERYCANCEL: // нажата кнопка закрытия окна свойств
          if ( ClickCancel ( hDlg) == false) // нажата кнопка отмены
          {
            // устанавливаем возвращаемое значение
            SetWindowLong (hDlg, DWL_MSGRESULT, TRUE);
          }
          else // завершаем программу
          {
            PostQuitMessage ( 0);
```

```
            return FALSE;
          }
          return TRUE;
      }
    }
      break;
    // обрабатываем сообщения кнопок
    case WM_COMMAND:
    {
     switch ( HIWORD ( wParam))
     {
       case BN_CLICKED: // нажата кнопка
           // устанавливаем состояние кнопки Next
           if ( LOWORD ( wParam) == IDC_YES)
              // включаем кнопку Next
              PropSheet_SetWizButtons ( GetParent ( hDlg), PSWIZB_BACK |
                                        PSWIZB_NEXT);
           else if ( LOWORD  (wParam) == IDC_NO)
              // выключаем кнопку Next
              PropSheet_SetWizButtons ( GetParent ( hDlg),
                                        PSWIZB_BACK);
           break;
     }
    }
      break;
    default:
      return FALSE;
   }
   return TRUE;
}
// функция загрузки текста лицензионного соглашения
int LoadLicense ( HWND hEdit)
{
   // ищем в ресурсах модуля программы текст лицензии
   HRSRC hrLic = FindResource ( GetModuleHandle ( NULL),
       MAKEINTRESOURCE ( IDR_LIC), TEXT ( "TXT_INFO"));
   if ( hrLic == NULL) // если не найден
        return -1; // выходим из функции
   // загружаем найденный ресурс в память
   HGLOBAL hGlobal = LoadResource ( GetModuleHandle  ( NULL), hrLic);
   if ( hGlobal == NULL) // если не удалось загрузить ресурс
         return -1; // выходим из функции
```

```
   // фиксируем ресурс в памяти
   LPVOID lpRsc = LockResource ( hGlobal);
   if ( lpRsc == NULL) // если ошибка
         return -1; // выходим из функции
   // выводим текст в окно редактирования
   if ( SendMessage ( hEdit, WM_SETTEXT, 0, ( LPARAM) lpRsc)== FALSE)
         return -1; // если ошибка, выходим из функции
   return 1;
}
// третье окно программы установки
BOOL APIENTRY InstallDlg ( HWND hDlg, UINT message, UINT wParam,
                           LONG lParam)
{
   // обработка сообщений
   switch ( message)
   {
   case WM_NOTIFY:
    {
      LPNMHDR lpnm = ( LPNMHDR) lParam;
      // обрабатываем сообщения страницы свойств
      switch ( lpnm->code)
      {
       case PSN_SETACTIVE: // инициализация страницы свойств
          // делаем активными кнопки Back и Finish
          PropSheet_SetWizButtons ( GetParent ( hDlg), PSWIZB_BACK |
                                    PSWIZB_FINISH);
          break;
       case PSN_WIZFINISH: // обрабатываем нажатие на кнопку Finish
           // устанавливаем программу
           SetupProgram ();
           // завершаем программу установки
           PostQuitMessage ( 0);
           return FALSE;
       case PSN_QUERYCANCEL: // нажата кнопка закрытия окна свойств
          if ( ClickCancel ( hDlg) == false) // нажата кнопка отмены
          {
            // устанавливаем возвращаемое значение
            SetWindowLong (hDlg, DWL_MSGRESULT, TRUE);
          }
          else // завершаем программу
          {
            PostQuitMessage ( 0);
            return FALSE;
          }
```

```
            return TRUE;
          )
        }
          break;
        default:
          return FALSE;
      }
      return TRUE;
}
```

Вот у нас и получилась полноценная заготовка для инсталляции любой программы на компьютер. Функция SetupProgram приведена условно для имитации процесса установки. Ее вид целиком зависит от программиста и может содержать функции распаковки и копирования файлов, создание ярлыков и регистрацию в системном реестре. В любом случае, ставилась задача изучения работы с окном свойств, и она успешно выполнена.

На этом я, к сожалению, должен завершить рассмотрение элементов управления. Многое осталось не сказанным, но даже приведенный в этой главе материал существенно поможет вам в повседневной работе.

ГЛАВА 19

Программирование оболочки

Думаю, читатель со мной согласится, что Windows, несмотря на все свои недостатки, является достаточно сложной и многокомпонентной операционной системой. Естественно, для поддержания ее в рабочем состоянии необходимы специальные модули и библиотеки, которые в свою очередь содержат многочисленные функции и интерфейсы. Было бы странным, если бы создатели этой операционной системы не позаботились о предоставлении программных средств доступа к ее широким возможностям. На данный момент фирма Microsoft предлагает разработчикам десятки функций и интерфейсов, позволяющих обращаться напрямую к системе, а также пользоваться ее внутренними сервисами и компонентами. Поскольку программирование оболочки является составляющей частью интерфейса Win32 API, мы рассмотрим в этой главе некоторые особенно интересные и полезные инструменты.

19.1. Функция *ShellExecute*

Сначала я хотел бы поговорить о наиболее часто применяемой в программах функции `ShellExecute`. Она позволяет открывать, просматривать, редактировать или выводить на печать различные зарегистрированные (имеющие запись в системном реестре) в Windows типы файлов. Фокус универсальности данной функции предельно прост: извлекается необходимая информация из реестра и загружается программа, поддерживающая указанный тип файлов. Здесь же и кроется один серьезный недостаток, который связан с выводом на печать выбранного файла. Не каждая программа содержит в себе код работы с принтером, и, следовательно, она не сможет распечатать файл, несмотря на то, что зарегистрирована в системе для работы с этим файлом.

Эта функция имеет шесть аргументов, часть которых обычно не нужна. Первый аргумент определяет дескриптор родительского окна. Второй аргу-

мент указывает на тип выполняемой операции (открыть, редактировать, просмотреть, печатать). Третий позволяет назначить имя файла или папки, для которой будет выполнена требуемая операция. Четвертый аргумент помогает задать параметры командной строки для выполняемого файла (например, EXE). Пятый помогает определить имя каталога по умолчанию, а шестой управляет режимом отображения окна (свернутое, развернутое скрытое). Для успешной компиляции программы подключите к проекту файл shellapi.h. Примеры работы с функцией `ShellExecute` показаны в листинге 19.1.

Листинг 19.1. Использование функции `ShellExecute`

```
#include <shellapi.h>
// открыть страницу в сети Интернет
ShellExecute ( NULL, "open", "http://www.MyWebPage.com", NULL, NULL,
               SW_SHOWNORMAL);
// открыть текстовый файл, расположенный в текущей папке программы
ShellExecute ( NULL, "open", "Readmy.txt", NULL, NULL, SW_SHOWNORMAL);
// вывести на печать файл Word из текущей папки
ShellExecute ( NULL, "print", "Document.doc", NULL, NULL,
               SW_SHOWMINIMIZED);
// найти папку Windows на корневом диске
ShellExecute ( NULL, "find", "c:\\Windows", NULL, NULL, SW_HIDE);
// просмотреть папку Windows
ShellExecute ( NULL, "explore", "c:\\windows", NULL, NULL,
               SW_SHOWNORMAL);
// написать письмо
LPCTSTR lpMail = "mailto:MyBox@service.com?subject=Anekdot \
                  &cc=MyBox@service.com";
ShellExecute ( NULL, NULL, lpMail, NULL, NULL, SW_SHOWNORMAL);
// запустить программу из текущей папки
ShellExecute ( NULL, NULL, "MyProgramm.exe", "-mycommand", NULL,
               SW_SHOWNORMAL);
// просмотреть графический файл
ShellExecute ( NULL, "open", "d:\\images\\pricol.bmp", NULL, NULL,
               SW_SHOWNORMAL);
```

Кроме основной функции `ShellExecute`, существует ее расширенный вариант, именуемый `ShellExecuteEx`. Отличается она, прежде всего, дополнительной поддержкой свойств объектов COM. Рассматривать ее мы не будем, поскольку в большинстве случаев вполне достаточно возможностей базовой функции.

19.2. Диалог выбора каталога

Следующим шагом изучения оболочки будет программирование системного диалога, позволяющего просматривать структуру каталогов и выбирать путь к определенной папке. Данный диалог идеально подходит для установки пути, используемого в дисковых операциях (например, сохранение файла в указанную пользователем папку). Чтобы облегчить себе задачу в дальнейшем, построим класс CDirView, поддерживающий диалоговое окно просмотра каталогов. Пример реализации этого класса показан в листингах 19.2 и 19.3.

Листинг 19.2. Файл DirView.h

```
#include <shlobj.h>
// объявление класса
class CDirView
{
public:
   CDirView (); // конструктор
   ~CDirView () { } // пустой деструктор
   // общая функция для вывода диалога на экран
   bool ShowBrowse ( HWND hParent, char* path, const char* title,
                     int iFolder);
}; // окончание класса
```

Листинг 19.3. Файл DirView.cpp

```
#include "stdafx.h"
#include "DirView.h"
// реализация класса CDirView
CDirView :: CDirView () // пустой конструктор
{
}
// функция вывода диалогового окна
bool CDirView :: ShowBrowse ( HWND hParent, char* path,
                              const char* title, int iFolder)
{
   LPMALLOC pMemory;
   // получаем указатель на интерфейс IMalloc
   if ( SUCCEEDED ( SHGetMalloc ( &pMemory)))
   {
     // указатель на структуру идентификатора для найденного каталога
     LPITEMIDLIST pitemDest = NULL;
```

```
// указатель на структуру идентификатора для исходной папки
LPITEMIDLIST pitemRoot = NULL;
// результат операции
bool bResult = false;
// временный буфер
TCHAR tmp[MAX_PATH + 1];
// обнуляем буфер
ZeroMemory ( tmp, MAX_PATH + 1);
// если задана начальная папка, получаем ее идентификатор
if ( iFolder)
    SHGetSpecialFolderLocation ( hParent, iFolder, &pitemRoot);
// определяем параметры структуры BROWSEINFO
BROWSEINFO browse;
// обнуляем структуру перед использованием
ZeroMemory ( &browse, sizeof ( BROWSEINFO));
// заполняем поля структуры
browse.hwndOwner = hParent; // дескриптор родительского окна
browse.lpszTitle = ( LPCTSTR) title; // текст заголовка
browse.pidlRoot = pitemRoot; // идентификатор исходной папки
browse.pszDisplayName = tmp; // буфер для хранения пути
// флаги для управления диалогом
browse.ulFlags = BIF_DONTGOBELOWDOMAIN | BIF_RETURNONLYFSDIRS |
                 BIF_RETURNFSANCESTORS;
// вызываем функцию оболочки для просмотра каталогов
pitemDest = SHBrowseForFolder ( &browse);
// если нажата кнопка Cancel, выходим из функции
bResult = ( pitemDest != NULL);
if ( bResult)
{
  // если нажата кнопка OK, получаем путь к выбранной папке
  try
  {
    SHGetPathFromIDList ( pitemDest, path);
  }
  catch ( . . .)
 {
    bResult = false;
 }
 // освобождаем ресурсы
 pMemory ->Free ( pitemDest);
}
// освобождаем выделенную память
pMemory->Release ();
```

```
    return bResult;
  }
  return false; // ошибка
}
```

Для реализации класса CDirView мы задействовали несколько функций оболочки, а также интерфейс IMalloc, позволяющий правильно распределить системную память. Основной функцией является SHBrowseForFolder, принимающая в качестве единственного аргумента указатель на структуру BROWSEINFO. В структуре определяются все основные параметры диалогового окна. Если задана начальная папка, с которой начнется просмотр (в нашем случае можно задавать только системные папки), вызывается функция SHGetSpecialFolderLocation. Она получает специальный идентификатор папки (CSIDL). Полный список существующих идентификаторов можно найти в документации к Win32 API. После того как пользователь сделал свой выбор и нажал кнопку **OK**, применяем функцию SHGetPathFromIDList, возвращающую путь к выбранной папке. Как видите, все довольно просто. Посмотрите примеры работы с классом CDirView, представленные в листинге 19.4.

Листинг 19.4. Использование класса CDirView

```
#include <shlobj.h>
// переменная для хранения пути к папке
char path[MAX_PATH];
CDirView dir;
// открываем окно по умолчанию
dir.ShowBrowse ( NULL, path, "Select Folder", 0);
// выбираем только из доступных логических дисков
dir.ShowBrowse ( NULL, path, "Select Folder", CSIDL_DRIVES);
// выбираем только из доступных сетевых дисков
dir.ShowBrowse ( NULL, path, "Select NetFolder", CSIDL_NETWORK);
```

Для того чтобы открывать не только системные, но и любые доступные на компьютере папки, придется немного усложнить класс CDirView. Во-первых, необходимо будет добавить обработку системных сообщений для диалогового окна просмотра каталогов, а во-вторых — добавить открытую функцию, задающую требуемый путь. Как это сделать, показано в листингах 19.5 и 19.6.

Листинг 19.5. Файл DirView.h для расширенного класса CDirView

```
#include <shlobj.h>
// объявление класса
class CDirView
```

```
{
public:
   CDirView (); // конструктор
   ~CDirView () { } // пустой деструктор
   // общая функция для вывода диалога на экран
   bool ShowBrowse ( HWND hParent, char* path, const char* title,
                     int iFolder);
   // расширенная функция вывода диалогового окна
   bool ShowAnyBrowse ( HWND hParent, char* path, const char* title,
                        bool bDefaultPath);
   // функция для установки пути к открываемой папке
   void SetFolder ( const char* Path);
private:
   // функция для обработки события BFFM_INITIALIZED
   void Initialize () const;
   // функция для обработки события BFFM_SELCHANGED
   void SelChanged ( const LPITEMIDLIST pidl) const;
   // статическая функция обратного вызова
   static int __stdcall BrowseCall (HWND hwnd, UINT uMsg, LPARAM lParam,
                                    LPARAM lpData);
   // идентификатор родительского окна
   HWND m_hWnd;
   // текущий выбранный путь к папке
   char m_szSelDir[MAX_PATH];
   // указатель на использование текущего пути к самой программе
   bool m_bDefaultPath;
}; // окончание класса
```

Листинг 19.6. Файл DirView.cpp для расширенного класса `CDirView`

```
#include "stdafx.h"
#include "DirView.h"
// реализация класса CDirView
CDirView :: CDirView () // пустой конструктор
{
   m_hWnd = NULL;
   m_bDefaultPath = false;
   strcpy( m_szSelDir, " ");
}
// функция для установки текущего пути
void CDirView :: SetFolder ( const char* Path)
{
   // если строка не содержит путь, выходим
   if ( strlen ( Path) < 3) return;
```

```
   // иначе сохраняем указанный путь
     strcpy ( m_szSelDir, Path);
}
// функция вывода диалогового окна
bool CDirView :: ShowBrowse ( HWND hParent, char* path,
                              const char* title, int iFolder)
{
   LPMALLOC pMemory;
   // получаем указатель на интерфейс IMalloc
   if ( SUCCEEDED ( SHGetMalloc ( &pMemory)))
   {
     // указатель на структуру идентификатора для найденного каталога
     LPITEMIDLIST pitemDest = NULL;
     // указатель на структуру идентификатора для исходной папки
     LPITEMIDLIST pitemRoot = NULL;
     // результат операции
     bool bResult = false;
     // временный буфер
     TCHAR tmp[MAX_PATH + 1];
     // обнуляем буфер
     ZeroMemory ( tmp, MAX_PATH + 1);
     // если задана начальная папка, получаем ее идентификатор
     if ( iFolder)
         SHGetSpecialFolderLocation ( hParent, iFolder, &pitemRoot);
     // определяем параметры структуры BROWSEINFO
     BROWSEINFO browse;
     // обнуляем структуру перед использованием
     ZeroMemory ( &browse, sizeof ( BROWSEINFO));
     // заполняем поля структуры
     browse.hwndOwner = hParent; // дескриптор родительского окна
     browse.lpszTitle = ( LPCTSTR) title; // текст заголовка
     browse.pidlRoot = pitemRoot; // идентификатор исходной папки
     browse.pszDisplayName = tmp; // буфер для хранения пути
     // флаги для управления диалогом
     browse.ulFlags = BIF_DONTGOBELOWDOMAIN | BIF_RETURNONLYFSDIRS |
                      BIF_RETURNFSANCESTORS;
     // вызываем функцию оболочки для просмотра каталогов
     pitemDest = SHBrowseForFolder ( &browse);
     // если нажата кнопка Cancel, выходим из функции
     bResult = ( pitemDest != NULL);
     if ( bResult)
     {
       // если нажата кнопка OK, получаем путь к выбранной папке
       try
```

```
      {
        SHGetPathFromIDList ( pitemDest, path);
      }
      catch ( . . .)
     {
        bResult = false;
     }
     // освобождаем ресурсы
     pMemory ->Free ( pitemDest);
    }
    // освобождаем выделенную память
    pMemory->Release ();
    return bResult;
  }
  return false; // ошибка
}
// расширенная функция вывода диалогового окна
bool CDirView :: ShowAnyBrowse ( HWND hParent, char* path,
                                  const char* title, bool bDefaultPath)
{
   bool bResult = false;
   LPITEMIDLIST pidl;
   BROWSEINFO browse;
   // обнуляем структуру перед использованием
   ZeroMemory ( &browse, sizeof ( BROWSEINFO));
   // заполняем поля структуры
   browse.hwndOwner = hParent; // дескриптор родительского окна
   browse.lpszTitle = ( LPCTSTR) title; // текст заголовка
   browse.pidlRoot = 0; // идентификатор исходной папки
   browse.pszDisplayName = 0; // буфер для хранения пути
   browse.ulFlags = BIF_RETURNONLYFSDIRS | BIF_STATUSTEXT; // флаги
   // функция обратного вызова
   browse.lpfn = BrowseCall;
   // указатель на класс CDirView
   browse.lParam = reinterpret_cast<long> ( this);
   // обрабатываем путь по умолчанию
   if ( bDefaultPath) m_bDefaultPath = true;
   // выводим на экран диалоговое окно
   if ( ( pidl = SHBrowseForFolder ( &browse)) != NULL)
   {
      // если нет ошибок, получаем путь к папке
      if ( SHGetPathFromIDList ( pidl, m_szSelDir))
      {
        bResult = true;
```

```
        strcpy ( path, m_szSelDir);
      }
      // если необходимо, освобождаем память
      LPMALLOC pMemory;
      if ( SUCCEEDED ( SHGetMalloc ( &pMemory)))
      {
        pMemory->Free ( pidl);
        pMemory->Release ();
      }
   }
   m_hWnd = NULL;
   // выходим из функции
   return bResult;
}
// функция для обработки события BFFM_SELCHANGED
void CDirView :: SelChanged ( const LPITEMIDLIST pidl) const
{
   TCHAR szCurDir[MAX_PATH];
   // получаем текущий выбранный путь
   if ( SHGetPathFromIDList ( ( LPITEMIDLIST) pidl,
                              ( LPTSTR) szCurDir))
   {
     // меняем текст в диалоговом окне
     SendMessage ( m_hWnd, BFFM_SETSTATUSTEXT, 0, ( LPARAM)  szCurDir);
   }
}
// функция для обработки события BFFM_INITIALIZED
void CDirView :: Initialize () const
{
   if ( m_bDefaultPath) // текущий путь
   {
     TCHAR szDir[MAX_PATH];
     // получаем путь к текущей папке
     if ( GetCurrentDirectory ( sizeof ( szDir) / sizeof ( TCHAR),
                                 szDir))
     {
       // посылаем сообщение диалоговому окну
       SendMessage ( m_hWnd, BFFM_SETSELECTION, TRUE, ( LPARAM) szDir);
     }
   }
   else // выбираем путь с помощью функции SetFolder
     SendMessage ( m_hWnd, BFFM_SETSELECTION, TRUE,
                   ( LPARAM) m_szSelDir);
}
```

Теперь наш класс позволяет указать любую доступную на компьютере (или в сети) папку, с которой начнется просмотр. Пример работы класса показан в листинге 19.7.

Листинг 19.7. Использование расширенного класса CDirView

```
#include <shlobj.h>
// переменная для хранения пути к папке
char path[MAX_PATH];
CDirView dir;
// начинаем просмотр с текущей папки
dir.ShowAnyBrowse ( NULL, path, "Select Folder", true);
// указываем начальную папку, с которой начнется просмотр
dir.SetFolder ( "C:\\Windows");
dir.ShowAnyBrowse ( NULL, path, "Select Folder", false);
```

Думаю, здесь все понятно, и можно переходить к следующей теме.

19.3. Диалог для открытия файлов

Одним из незаменимых компонентов оболочки является набор стандартных диалоговых окон, решающих следующие задачи: открытие и сохранение файлов, выбор шрифта и цвета, поиск и замена текста. Программисту предоставляются три варианта работы с этими диалогами. Первый позволяет просто воспользоваться готовым решением, второй дает возможность получить доступ к внутренней структуре этих ресурсов, а третий вариант полностью заменяет стандартный ресурс пользовательским. Как правило, для большинства задач вполне хватает первого варианта, однако если вам необходимо реализовать нестандартные возможности, придется применить второй или третий варианты.

Сначала рассмотрим, как можно использовать базовые ресурсы системы на примере диалога "**Открыть**". Создайте класс COpenFile, как показано в листингах 19.8 и 19.9.

Листинг 19.8. Файл OpenFile.h класса COpenFile

```
#include <Commdlg.h>
#define CH_BUFFER_SIZE  5120 // размер буфера
#define OP_NAMESIZELEN  100 // длина имени файла
// объявление класса
class COpenFile
```

```
{
public:
   // конструктор
   COpenFile ( HWND hMain, char* szFilter, bool bMulti);
   // деструктор
   ~CopenFile ();
// общие функции
   // показать диалог для открытия файла
   void* ShowOpenDlg ();
   // показать диалог для сохранения файла
   void* ShowSaveDlg ();
   // получить путь к файлу
   void GetFilePath ( char* szPath) const;
   // получить имя файла
   void GetFileName ( char* szName) const;
   // получить имя первого файла
   int GetFirstFile ( char* szFile);
   // получить имя следующего файла
   int GetNextFile ( char* szFile);
private:
   char m_szFilePath[MAX_PATH];
   char m_szFileName[OP_NAMESIZELEN];
   char* m_pszFile;
   char* m_pszPath;
   OPENFILENAME m_of;
   bool m_bMulti;
   unsigned int m_pos;
   unsigned int m_len_path;
   int m_len;
   bool m_bExpl;
}; // окончание класса
```

Листинг 19.9. Файл OpenFile.cpp класса `COpenFile`

```
#include "stdafx.h"
#include "OpenFile.h"
// реализация класса
COpenFile :: COpenFile ( HWND hMain, char *szFilter, bool bMulti)
{
   // инициализируем переменные
   m_szFilePath[0] = '\0';
   m_szFileName[0] = '\0';
   m_pszFile = NULL;
```

```
m_pszPath = NULL;
m_bMulti = bMulti;
m_pos = 0;
m_len_path = 0;
m_len = 0;
// если будет использоваться множественный выбор файлов, готовим буфер
if ( m_bMulti)
{
  // выделяем немного памяти для хранения имен файлов
  m_pszFile = new char[CH_BUFFER_SIZE]; // 1024 * 5
  // если память недоступна, выходим
  if ( m_pszFile == NULL) return;
  // и еще немного для хранения пути к файлам
  m_pszPath = new char[CH_BUFFER_SIZE];
  // если память недоступна, выходим
  if ( m_pszPath == NULL)
  {
    delete [] m_pszFile;
    return;
  }
  m_pszPath[0] = NULL;
}
// обнуляем файловую структуру
memset( &m_of, 0, sizeof ( OPENFILENAME));
// заполняем поля структуры
m_of.lStructSize = sizeof ( OPENFILENAME); // размер структуры
m_of.hwndOwner = hMain;  // дескриптор родительского окна
m_of.hInstance = 0;
m_of.lpstrFilter = szFilters; // фильтр для выбора типов файлов
m_of.lpstrCustomFilter = 0;
m_of.nMaxCustFilter = 0;
m_of.nFilterIndex = 0;
// если будет использоваться множественный выбор файлов
if ( m_bMulti)
{
  // используем выделенный буфер для хранения имен файлов
  m_of.lpstrFile = m_pszFile;
  m_of.lpstrFile[0] = NULL;
}
else
   m_of.lpstrFile = m_szFilePath;
// определяем размер выделенного буфера
if ( m_bMulti)
   m_of.nMaxFile = CH_BUFFER_SIZE + 1;
```

```
   else
      m_of.nMaxFile = sizeof ( m_szFilePath);
   // продолжаем
   m_of.lpstrFileTitle = m_szFileName; // заголовок диалогового окна
   m_of.nMaxFileTitle = sizeof ( m_szFileName);
   m_of.lpstrInitialDir = 0;
   m_of.lpstrTitle = 0;
   // определяем флаги
   if ( m_bMulti)
      m_of.Flags = OFN_ALLOWMULTISELECT | OFN_HIDEREADONLY |
                   OFN_FILEMUSTEXIST | OFN_EXPLORER;
   else
      m_of.Flags = OFN_HIDEREADONLY | OFN_FILEMUSTEXIST;
   // продолжаем
   m_of.nFileOffset = 0;
   m_of.nFileExtension = 0;
   m_of.lCustData = 0;
   m_of.lpfnHook = 0;
   m_of.lpTemplateName = 0;
}
COpenFile :: COpenFile ()
{
   // освобождаем память
   if ( m_pszFile) delete [] m_pszFile;
   if ( m_pszPath) delete [] m_pszPath;
}
void* COpenFile :: ShowOpenDlg ()
{
   return &m_of;
}
void* COpenFile :: ShowSaveDlg ()
{
   return &m_of;
}
void COpenFile :: GetFilePath ( char* szPath) const
{
   strcpy( szPath, m_szFilePath);
}
void COpenFile :: GetFileName ( char* szName) const
(
   strcpy ( szName, m_szFileName);
}
```

```
int COpenFile :: GetFirstFile (char* szFile)
{
   char buf[MAX_PATH] = "";
   // если выбран не множественный режим, выходим
   if ( !m_bMulti) return 0;
   // определяем длину буфера
   len = strlen ( m_pszFile);
   // проверяем стиль окна
   m_bExpl = m_of.Flags & OFN_EXPLORER;
   // если флаг OFN_EXPLORER установлен, делаем так
   if ( m_bExpl)
   {
     m_len = CH_BUFFER_SIZE;
     // выделяем общий путь к файлам
     strncpy ( m_pszPath, m_pszFile, strlen ( m_pszFile));
     m_pszPath[m_of.nFileOffset-1] = '\0';
     strcat ( m_pszPath, "\\");
     // получаем смещение к первому имени файла
     m_pos = m_of.nFileOffset - 1;
     m_pos++;
     m_len -= m_pos;
     // выделяем имя первого файла
     for ( int j = 0; j < m_len; j++)
     {
        if ( m_pszFile[m_pos] == '\0')
        {
          strcat ( m_pszPath, buf);
          strcpy ( szFile, m_pszPath);
          m_pszPath[m_of.nFileOffset] = '\0';
          m_len_path = m_of.nFileOffset;
          m_len += j;
          m_pos++;
          return 1;
        }
        buf[j] = m_pszFile[m_pos];
        m_pos++;
     }
   // если флаг OFN_EXPLORER не установлен, делаем так
   else
   {
     // если выбран один файл
     if ( m_of.nFileExtension)
```

```
    {
     // копируем путь и выходим
     strcpy ( szFile, m_of.lpstrFile);
     return 0;
    }
    else // иначе получаем имя первого файла
    {
     for ( int i = 0; i < m_len; i++)
     {
      if ( m_pszFile[i] == ' ')
      {
       m_pszPath[0] = '\0';
       strncpy ( m_pszPath, buf, strlen ( buf));
       m_pszPath[i] = '\0';
       buf[0] = '\0';
       m_pos = i;
       m_pos++;
       m_len -= i;
       for ( int j = 0; j < m_len; j++)
       {
         if ( m_pszFile[m_pos] == ' ')
         {
           buf[j] = '\0';
           strcpy ( szFile, m_pszPath);
           m_pos ++;
           m_pszPath[i] = '\0';
           m_len_path = i;
           m_len += i;
           return 1;
         }
         buf[j] = m_pszFile[m_pos];
         m_pos++;
       }
      }
      buf[i] = m_pszFile[i];
     }
    }
  }
  return 0;
}

int COpenFile :: GetNextFile ( char* szFile)
{
  char buf[MAX_PATH] = " ";
```

```
// устанавливаем ограничения
if ( !m_bMulti) return 0;
if ( strlen ( m_pszPath) < 3) return 0;
if ( ( m_pszFile[m_pos]) == '\0') return 2;
// если флаг OFN_EXPLORER установлен, делаем так
if( m_bExpl)
(
  // получаем имя следующего файла
  for ( int i = 0; i < m_len; i++)
  {
    if ( m_pos >= m_len) return 2;
    if ( m_pszFile[m_pos] == '\0')
    {
      if ( m_pszFile[m_pos + 1] == '\0')
      {
        strcat ( m_pszPath, buf);
        strcpy ( szFile, m_pszPath);
        return 2;
      }
      strcat ( m_pszPath, buf);
      strcpy ( szFile, m_pszPath);
      m_pos ++;
      m_pszPath[m_len_path] = '\0';
      return 1;
    }
    buf[i] = m_pszFile[m_pos];
    m_pos++;
  }
}
else
{
  for ( int i = 0; i < m_len; i++)
  {
    // получаем имя файла
    if ( m_pos >= m_len) return 2;
    if ( m_pszFile[m_pos] == ' ')
    {
     buf[i] = '\0';
     strcat ( m_pszPath, buf);
     strcpy ( szFile, m_pszPath);
     m_pos ++;
     m_pszPath[m_len_path] = '\0';
     return 1;
    }
```

```
        buf[i] = m_pszFile[m_pos];
        m_pos++;
      }
   }
   return 0;
}
```

Класс получился несколько сложным из-за того, что была добавлена возможность загрузки множества файлов, поскольку в некоторых программах (например, аудио- и видеоплееры) без этого просто не обойтись. Пример работы с классом `COpenFile` представлен в листинге 19.10. Для исключения неприятных ошибок при создании выполняемого файла добавьте в опциях компоновщика ссылку на библиотеку Comdlg32.lib.

Листинг 19.10. Пример работы с классом `COpenFile`

```
#include "OpenFile.h"
// определяем фильтр для выбора файлов
char szFilter[] = _T ( "All Files (*.*)\0*.*\0\0");
// буфер для хранения пути
char szPath [MAX_PATH] = "";
// объявляем наш класс с множественным выбором
COpenFile of ( hMyWindow, szFilter, true);
// выводим на экран диалог для открытия файлов
int iCheck = GetOpenFileName ( ( LPOPENFILENAME) of.ShowOpenDlg ());
// если пользователь выбрал более одного файла, обрабатываем их
if ( iCheck != 0)
{
   // получаем имя и путь для первого файла
   if ( of.GetFirstFile ( szPath))
   {
      // делаем что-нибудь с полученным значением
      // например, копируем в стандартный список
      SendMessage ( hMyList, LB_ADDSTRING, 0, ( LPARAM)
                  ( LPCTSTR) szPath);
      // выбираем оставшиеся файлы
      while ( 1)
      {
        if ( of.GetNextFile ( szPath) == 2)
        {
          if ( strlen ( szPath) > 5)
          SendMessage ( hMyList, LB_ADDSTRING, 0, ( LPARAM)
                      ( LPCTSTR) szPath);
```

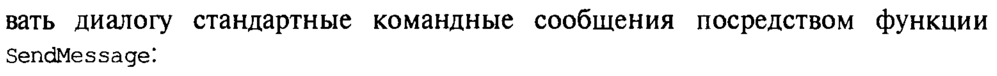
```
          return; // если файл последний, выходим
        }
        SendMessage ( hMyList, LB_ADDSTRING, 0, ( LPARAM)
                    ( LPCTSTR) szPath);
      )
    }
    else // выбран только один файл
    {
      // копируем его в список
      SendMessage ( hMyList, LB_ADDSTRING, 0, ( LPARAM)
                  ( LPCTSTR) szPath);
    }
}
```

В рассмотренном примере мы определили фильтр, позволяющий открывать любые файлы, и поддержку множественного выбора (более одного файла). После этого была вызвана системная функция GetOpenFileName для вывода на экран стандартного диалогового окна. Когда пользователь отметит открываемые файлы и нажмет кнопку **OK**, в буфер szPath будет помещено определенное значение. Если выбран только один файл, это значение будет содержать путь и имя файла, иначе придется применить функции GetFirstFile и GetNextFile для чтения из буфера всех имен файлов.

Чтобы получить доступ к внутренней структуре стандартных диалоговых окон, придется немного изменить класс COpenFile. Нам нужно будет определить собственную функцию обратного вызова CallOpenFileProc, в которой можно обрабатывать специальные системные сообщения для диалогов. Кроме того, необходимо добавить флаг OFN_ENABLEHOOK при определении структуры OPENFILENAME. Диалоговое окно для открытия (сохранения файла) поддерживает следующие уведомляющие сообщения:

- CDN_FILEOK — пользователь нажал кнопку **OK**;
- CDN_FOLDERCHANGE — пользователь открыл новую папку;
- CDN_HELP — пользователь нажал кнопку помощи;
- CDN_INITDONE — завершена инициализация диалогового окна;
- CDN_SELCHANGE — выбран новый файл в открытой папке;
- CDN_SHAREVIOLATION — пользователь нажал кнопку **OK** и произошла ошибка доступа к общему сетевому ресурсу;
- CDN_TYPECHANGE — пользователь выбрал новый тип файла в раскрывающемся списке.

Все эти уведомления можно обработать в функции обратного вызова через системное сообщение WM_NOTIFY. Также существует возможность переда-



вать диалогу стандартные командные сообщения посредством функции SendMessage:

- CDM_GETFILEPATH — получить имя и путь к выбранному файлу;
- CDM_GETFOLDERIDLIST — получить системный идентификатор текущей папки;
- CDM_GETFOLDERPATH — получить путь для текущей папки;
- CDM_HIDECONTROL — спрятать определенный элемент управления диалога;
- CDM_SETCONTROLTEXT — изменить текст для указанного элемента управления диалога;
- CDM_SETDEFEXT — установить расширение файла по умолчанию.

С их помощью можно не только обрабатывать выбранные файлы, но и преобразить внешний вид самого диалогового окна. Перед тем как приступить к практике, ознакомьтесь с идентификаторами элементов управления диалогового окна, показанных в табл. 19.1. С их помощью можно посылать командные сообщения для определенного элемента.

Таблица 19.1. *Идентификаторы элементов управления*

Идентификатор	Описание элемента управления
IDOK	Кнопка **OK**
IDCANCEL	Кнопка **Cancel**
pshHelp	Кнопка **Help**
chx1	Флажок "только для чтения"
cmb1	Раскрывающийся список для отображения типов файлов (определяется фильтром)
cmb2	Раскрывающийся список для отображения выбранного диска или папки
cmb13	Раскрывающийся список для редактирования имени открываемого файла
lst1	Список для отображения содержимого текущей папки или диска
edt1	Поле редактирования, отображающее имя выбранного файла с возможностью редактирования
stc1	Надпись для списка lst1
stc2	Надпись для раскрывающегося списка cmb1
stc3	Надпись для раскрывающегося списка cmb13 и поля редактирования edt1
stc4	Надпись для раскрывающегося списка cmb2

Теперь, когда основные теоретические сведения получены, перейдем к программированию диалогового окна с помощью функции обратного вызова. В листинге 19.11 показаны изменения, которые нужно внести в реализацию уже созданного класса `COpenFile`.

Листинг 19.11. Перехват внутренних сообщений для диалогового окна

```
// исправьте конструктор класса, как показано ниже
COpenFile ( HWND hMainDialog, char* szFilters, bool bMulti,
            bool bHook = false);
// добавьте в конец функции конструктора следующий код
if ( bHook) // добавляем флаг OFN_ENABLEHOOK
{
   if ( m_bMulti) // если  установлен множественный выбор
        ofn.Flags = OFN_ALLOWMULTISELECT | OFN_ENABLEHOOK | OFN_EXPLORER
                   | OFN_HIDEREADONLY;
   else
         ofn.Flags = OFN_ENABLEHOOK | OFN_EXPLORER | OFN_HIDEREADONLY;
         // добавляем указатель на функцию обратного вызова
         ofn.lpfnHook = ( LPOFNHOOKPROC) FileOpenHookProc;
}
else
   ofn.lpfnHook = 0;
// добавьте объявление глобальных функций в начало файла OpenFile.cpp
// функция обратного вызова для перехвата сообщений диалога
BOOL CALLBACK FileOpenHookProc ( HWND, UINT, WPARAM, LPARAM);
// функция обработки специальных уведомляющих сообщений
BOOL NEAR PASCAL FileOpenNotifyProc ( HWND hDlg, LPOFNOTIFY lpNotify);
// пишем реализацию этих функций
BOOL CALLBACK FileOpenHookProc ( HWND hDlg, UINT message, WPARAM wParam,
                                 LPARAM lParam)
{
   // обрабатываем стандартные сообщения Windows
   switch ( message)
   {
    case WM_INITDIALOG: // инициализация диалога
     // здесь можно устанавливать новый текст для элементов диалога
     // SendMessage ( hDlg, CDM_SETCONTROLTEXT, ( WPARAM) stc3,
                    ( LPARAM) "Имечко:");
     // или прятать отдельные злементы
     // SendMessage ( hDlg, CDM_HIDECONTROL, ( WPARAM} IDOK, 0);
     // если нам понадобится указатель на структуру OPENFILENAME
     SetWindowLong ( hDlg, DWL_USER, lParam);
     break;
```

```
    // если нужно, может изменить цвет диалогового окна
    // case WM_CTLCOLORDLG:
    // создать кисть следует заранее
    // return ( LRESULT) m_Brush;
    // обработка уведомляющих сообщений диалога
    case WM_NOTIFY:
     // вызываем функцию обработки сообщений
     FileOpenNotifyProc ( hDlg, ( LPOFNOTIFY) lParam);
     break;
    // закрытие окна
    case WM_DESTROY:
    {
     // если нужно, получаем указатель на структуру OPENFILENAME
     LPOPENFILENAME lpOf = ( LPOPENFILENAME) GetWindowLong ( hDlg,
                             DWL_USER);
    }
     break;
   }
   return FALSE;
}
// функция обработки уведомляющих сообщений диалогового окна
BOOL NEAR PASCAL FileOpenNotifyProc ( HWND hDlg, LPOFNOTIFY lpNotify)
{
   // определяем переменную для хранения данных
   char szTemp[MAX_PATH];
   // обрабатываем коды уведомляющих сообщений
   switch ( lpNotify->hdr.code)
   {
    case CDN_INITDONE: // инициализация диалога
      MessageBox ( hDlg, "Инициализация диалога.", "CDN_INITDONE",
                   MB_OK);
      break;
    case CDN_FILEOK: // нажата кнопка OK
      // выводим имя выбранного файла
      MessageBox ( hDlg, lpNotify->lpOFN->lpstrFile, "CDN_FILEOK",
                   MB_OK);
      break;
    case CDN_FOLDERCHANGE: // выбрана новая папка
    {
      // получаем путь к папке, используя макрос
      if ( CommDlg_OpenSave_GetFolderPath ( GetParent ( hDlg), szTemp,
                              sizeof ( szTemp)) <= sizeof(szTemp))
```

```
      {
        // отображаем имя папки
        MessageBox ( hDlg, szTemp, "CDN_FOLDERCHANGE", MB_OK);
      }
    }
      break;
    case CDN_SELCHANGE: // выбран новый файл
    {
      // выводим путь к файлу
      if ( CommDlg_OpenSave_GetFilePath ( GetParent ( hDlg), szTemp,
                              sizeof ( szTemp)) <= sizeof ( szTemp))
      {
        MessageBox ( hDlg, szTemp, "CDN_SELCHANGE", MB_OK);
      }
      // выводим имя файла
      if ( CommDlg_OpenSave_GetSpec ( GetParent ( hDlg), szTemp,
                           sizeof ( szTemp)) <= sizeof ( szTemp))
      (
         MessageBox ( hDlg, szTemp, "CDN_SELCHANGE", MB_OK);
      }
    }
      break;
    case CDN_SHAREVIOLATION: // ошибка доступа к файлу в сети
     MessageBox ( hDlg, lpNotify->lpOFN->lpstrFile, "Ошибка", MB_OK);
     break;
    case CDN_HELP: // нажата кнопка справки
     MessageBox ( hDlg, "Не удалось найти файл справки", " CDN_HELP ",
                 MB_OK);
     break;
   }
   return TRUE;
}
```

Как видно из листинга, реализовать перехват сообщений для диалогового окна довольно просто. Однако следует знать, что некоторые возможности (изменение цвета диалога, модификация текста элементов управления и их отображение) не будут поддерживаться, если установлен стиль OFN_EXPLORER.

Последний способ, связанный с созданием собственного диалогового окна, мы здесь рассматривать не будем, поскольку он имеет много общего с уже приведенными способами и, думаю, не создаст проблем опытному программисту. А я хотел бы рассказать о не менее важном компоненте оболочки — о ярлыке.

19.4. Создание ярлыка

Если говорить на языке программирования, ярлык можно представить как ссылку на какой-либо объект операционной системы (например, на файл), расположенный в любом месте вашего компьютера (на одном из дисков). При щелчке мышью по ярлыку Windows проверяет его свойства и активизирует соответствующую процедуру обработки (например, запуск программы на выполнение). Как правило, большая часть ярлыков сосредоточена на Рабочем столе и в меню кнопки **Пуск**, поскольку именно отсюда пользователь начинает работать с оболочкой. Несомненно, любому программисту рано или поздно понадобится программно создавать и размещать ярлыки как на Рабочем столе, так и в различных каталогах на компьютере. Мы рассмотрим сам процесс создания ярлыка и несколько практических примеров его размещения на компьютере.

Для поддержки ярлыков в Windows существует специальный интерфейс `IShellLink`, с помощью которого можно легко манипулировать свойствами ярлыков. Также нам понадобится вспомогательный интерфейс `IPersistFile`, позволяющий загружать и сохранять различные файлы на диск. Думаю, читатель знает, что интерфейс не является классом и его нельзя вызвать напрямую. Сначала нужно инициализировать библиотеку COM-объектов (функция `CoInitialize`) и получить глобальный идентификатор (`CLSID`) для используемого интерфейса (функция `CoCreateInstance`). После этого можно пользоваться всеми доступными методами интерфейса, а по окончании работы нужно обязательно освободить объектную модель COM (функция `CoUninitialize`). По традиции напишем специальный класс для создания ярлыков и назовем его, например, `CLink`. Необходимые файлы представлены в листингах 19.12 и 19.13.

Листинг 19.12. Файл Link.h класса `CLink`

```
#include <shlobj.h>
// определяем класс CLink
class CLink
{
public:
   Clink (); // конструктор
   ~Clink () { } // пустой деструктор
// общие функции класса
   // функция создания ярлыка на Рабочем столе
   void CreateLnkDesktop ( char* pszNameLinkFile, char* pszPathname,
                           char* pszArgs, char* pszIconPath, int iIcon);
   // функция создания ярлыка для самой программы
   void CreateLnkThis ( char* NameProgram, char* NameLnk,
                        DWORD dwFolder);
```

```
private:
   HRESULT hres; // возвращаемое значение
   IShellLink* psl; // поддержка ярлыков
   IPersistFile* ppf; // поддержка файловых операций
   TCHAR lpszSpecialFolderPath[MAX_PATH]; // путь к системным папкам
   WCHAR wsz [MAX_PATH]; // временный буфер
}; // окончание класса
```

Листинг 19.13. Файл Link.cpp класса `CLink`

```
#include "stdafx.h"
#include "Link.h"
// реализация класса
CLink :: Clink ()
{
   hres = 0;
   psl = 0;
   ppf = 0;
}
void CLink :: CreateLnkDesktop ( char* pszNameLinkFile,
         char* pszPathname, char* pszArgs, char* pszIconPath, int iIcon)
{
   TCHAR pszLinkFile[MAX_PATH];
   // инициализируем COM
   hres = CoInitialize ( NULL);
   // проверяем результат
   if ( hRes != S_OK) return;
   // получаем идентификатор интерфейса IshellLink и создаем объект
   hres = CoCreateInstance ( CLSID_ShellLink, NULL, CLSCTX_INPROC_SERVER,
                              IID_IShellLink, ( PVOID*) &psl);
   // проверяем результат
   if ( SUCCEEDED ( hres))
   {
     // получаем путь к папке Рабочий стол
     if ( SHGetSpecialFolderPath ( 0, lpszSpecialFolderPath,
                                    CSIDL_DESKTOP, FALSE))
     {
       // сохраняем путь к системной папке
       strcpy ( pszLinkFile, lpszSpecialFolderPath);
       // создаем путь для ярлыка
       strcat ( pszLinkFile, "\\");
       strcat ( pszLinkFile, pszNameLinkFile);
```

```
      // определяем путь для программы, на которую указывает ярлык
      psl->SetPath ( pszPathname);
      // если программе нужны аргументы, добавляем их
      psl->SetArguments ( pszArgs);
      // определяем путь к значку для ярлыка
      psl->SetIconLocation ( pszIconPath, iIcon);
      // получаем указатель на интерфейс IPersistFile
      hres = psl->QueryInterface ( IID_IPersistFile, ( PVOID*) &ppf);
      // сохраняем файл ярлыка на диск
      if ( SUCCEEDED ( hres))
      {
        // если кодировка Unicod, преобразуем строку
        #ifndef UNICODE
        MultiByteToWideChar ( CP_ACP, 0, pszLinkFile, -1, wsz,
                              MAX_PATH);
        // сохраняем ярлык
        hres = ppf->Save ( wsz, TRUE);
        #else
        hres = ppf->Save ( pszLinkFile,TRUE);
        #endif
        // освобождаем интерфейс IPersistFile
        ppf->Release ();
      }
    }
    // освобождаем интерфейс IShellLink
    psl->Release ();
  }
  // удаляем COM
  CoUninitialize ();
  // обнуляем переменные
  hres = 0;
  psl = 0;
  ppf = 0;
}
void CLink :: CreateLnkThis ( char* NameProgram, char* NameLnk,
                              DWORD dwFolder)
{
  char pszLinkFile[MAX_PATH]; // путь к ярлыку
  char pszPathname[MAX_PATH]; // путь к программе
  char pszIconPath[MAX_PATH]; // путь к значку для ярлыка
  // получаем полный путь к своей программе
  GetModuleFileName ( NULL, pszPathname, MAX_PATH);
```

```
// восстанавливаем короткий вариант пути, если необходимо
if ( GetShortPathName ( pszPathname, szShort, MAX_PATH))
{
  strcpy ( pszPathname, szShort);
}
// убираем из пути имя программы
*( strrchr ( pszPathname, '\\') + 1) = '\0';
// создаем путь к ярлыку
strcpy ( pszIconPath, pszPathname);
strcat ( pszIconPath, "\\");
strcat ( pszIconPath, NameProgram);
// инициализируем COM
hres = CoInitialize ( NULL);
// проверяем результат
if ( hRes != S_OK) return;
// получаем идентификатор интерфейса IshellLink и создаем объект
hres = CoCreateInstance ( CLSID_ShellLink, NULL, CLSCTX_INPROC_SERVER,
                          IID_IShellLink, ( PVOID*) &psl);

// проверяем результат
if ( SUCCEEDED ( hres))
{
  // получаем путь к указанной папке
  if ( SHGetSpecialFolderPath ( 0, lpszSpecialFolderPath,
                                dwFolder, FALSE))
  {
    // копируем путь
    strcpy ( pszLinkFile, lpszSpecialFolderPath);
    strcat ( pszLinkFile, "\\");
    // добавляем имя ярлыка
    strcat ( pszLinkFile, NameLnk);
    // устанавливаем рабочий каталог
    psl->SetWorkingDirectory ( pszPathname);
    // устанавливаем путь к ярлыку
    strcpy ( pszPathname, pszIconPath);
    psl->SetPath ( pszPathname);
    // устанавливаем путь к значку
    psl->SetIconLocation ( pszIconPath, 0);
    // сохраняем ярлык на диск
    hres = psl->QueryInterface ( IID_IPersistFile, ( PVOID*) &ppf);
    if ( SUCCEEDED ( hres))
    {
      MultiByteToWideChar ( CP_ACP, 0, pszLinkFile, -1, wsz,
                            MAX_PATH);
```

```
            hres = ppf->Save ( wsz, TRUE);
            ppf->Release ();
         }
         psl->Release ();
      }
   }
   // удаляем COM
   CoUninitialize ();
   // обнуляем переменные
   hres = 0;
   psl = 0;
   ppf = 0;
}
```

В классе мы определили две функции: `CreateLnkDesktop` и `CreateLnkThis`. Первая позволяет создать любой ярлык и поместить его на Рабочий стол, а вторая предназначена для создания ярлыка к самой выполняемой программе и размещения его в любой системной папке. Примеры работы с классом `CLink` показаны в листинге 19.14.

Листинг 19.14. Примеры работы с классом CLink

```
// создание ярлыка для перезагрузки компьютера
void Restart_Windows ()
{
   CLink link;
   TCHAR lpszWin[MAX_PATH];
   // получаем путь к папке с Windows
   if ( GetWindowsDirectory ( lpszWin, MAX_PATH))
   {
     // создаем ярлык
     strcat ( lpszWin, "\\rundll32.exe");
     link.CreateLnkDesktop ( "RestartWindows.lnk", lpszWin,
            "shell32.dll, SHExitWindowsEx 0x2", "shell32.dll", -153);
   }
}
// создание ярлыка для отображения свойств экрана
void ShowDisplayProperty ()
{
   CLink link;
   TCHAR lpszWin[MAX_PATH];
   // получаем путь к папке с Windows
   if ( GetWindowsDirectory ( lpszWin, MAX_PATH))
```

```
   {
     // создаем ярлык
     strcat ( lpszWin, "\\rundll32.exe");
     link.CreateLnkDesktop ( "Display Properties.lnk", lpszWin,
   "shell32.dll,Control_RunDLL desk.cpl,Display,3", "shell32.dll", -43);
   }
}
// функция для создания ярлыка программы на Рабочем столе
void CreateShortcutDesktop ()
{
   CLink link;
   // создаем ярлык
   link.CreateLnkThis ( "MyProgram.exe", "MyProgram.lnk",
                        CSIDL_DESKTOP);
}
// функция для создания ярлыка программы в меню кнопки Пуск
void CreateShortcutStartMenu ()
{
   CLink link;
   // создаем ярлык
   link.CreateLnkThis ( "MyProgram.exe", "MyProgram.lnk",
                        CSIDL_STARTMENU);
}
```

Думаю, что представленных примеров будет достаточно для самостоятельного продолжения работы с классом CLink.

19.5. Процессы и потоки

И напоследок поговорим о доступе к потокам и процессам в Windows. Самым простым способом получения информации о текущих процессах и потоках является использование функции CreateToolhelp32Snapshot. Она позволяет сделать снимок текущего состояния системы. Дополнительно к ней применяются функции Process32First, Process32Next, Thread32First и Thread32Next. Первые две функции позволяют найти и извлечь информацию о процессах, а последние две выполняют те же функции, но для потоков. Для описания процессов и потоков используются соответственно структуры PROCESSENTRY32 и THREADENTRY32. Пример получения информации обо всех текущих процессах и потоках показан в листинге 19.15.

Листинг 19.15. Получение информации о запущенных процессах и потоках

```
#include "stdafx.h"
#include <stdio.h>
#include <tlhelp32.h>
```

```
// пишем функцию, загружающую найденные процессы в список
void LoadProcess ( HWND hListBox)
{
   // определяем информационную структуру для процесса
   PROCESSENTRY32 procEntry = { 0 };
   // буфер для данных
   TCHAR buffer[100] = " ";
   // условие завершения поиска
   bool bEnd = false;
   // делаем "снимок" системы
   HANDLE hSnapshot = CreateToolhelp32Snapshot ( TH32CS_SNAPPROCESS, 0);
   // заносим размер структуры PROCESSENTRY32
   procEntry.dwSize = sizeof ( PROCESSENTRY32);
   // получаем первый найденный процесс
   bEnd = Process32First ( hSnapshot, &procEntry};
   // создаем цикл для перечисления всех загруженных процессов
   while ( bEnd)
   {
     // получаем имя выполняемого файла для найденного процесса
     sprintf ( buffer, "File: %s", procEntry.szExeFile);
     // заносим его в список
     SendMessage ( hListBox, LB_ADDSTRING, 0,
                  ( LPARAM) ( LPCTSTR) buffer);
     // получаем уникальный идентификатор процесса
     sprintf ( buffer, "ID: %5X", procEntry.th32ProcessID);
     // заносим его в список
     SendMessage ( hListBox, LB_ADDSTRING, 0,
                  ( LPARAM) ( LPCTSTR) buffer);
     // получаем значение базового приоритета
     sprintf ( buffer, "Priority: %ld", procEntry.pcPriClassBase);
     // заносим его в список
     SendMessage ( hListBox, LB_ADDSTRING, 0,
                  ( LPARAM) ( LPCTSTR) buffer);
     // получаем число потоков в процессе
     sprintf ( buffer, "Count Threads: %ld", procEntry.cntThreads);
     // заносим его в список
     SendMessage ( hListBox, LB_ADDSTRING, 0,
                  ( LPARAM) ( LPCTSTR) buffer);
     // ищем потоки для заданного процесса
     LoadThread ( hListBox, procEntry.th32ProcessID);
     // переходим к поиску следующего процесса
     bEnd = Process32Next ( hSnapshot, &procEntry);
   }
```

```
   // закрываем дескриптор поиска
   CloseHandle ( hSnapshot);
}
// пишем функцию для получения потоков для указанного процесса
void LoadThread ( HWND hListBox, DWORD dwProcessID)
{
   // определяем информационную структуру для процесса
   THREADENTRY32 threadEntry = { 0 };
   // буфер для данных
   TCHAR buffer[100] = " ";
   // условие завершения поиска
   bool bEnd = false;
   // счетчик потоков
   int iNumber = 1;
   // делаем "снимок" системы
   HANDLE hSnapshot = CreateToolhelp32Snapshot ( TH32CS_SNAPTHREAD, 0);
   // заносим размер структуры THREADENTRY32
   threadEntry.dwSize = sizeof ( THREADENTRY32);
   // получаем первый найденный поток
   bEnd = Thread32First ( hSnapshot, &threadEntry);
   // создаем цикл для перечисления всех потоков
   while ( bEnd)
   {
     // проверяем идентификатор процесса
     if ( threadEntry.th32OwnerProcessID == dwProcessID)
     {
      // получаем идентификатор потока
      sprintf ( buffer, "Thread %u, ID: %5X", iNumber,
                threadEntry.th32ThreadID);
      // заносим его в список
      SendMessage ( hListBox, LB_ADDSTRING, 0,
                    ( LPARAM) ( LPCTSTR) buffer);
      // получаем базовый приоритет потока
      sprintf ( buffer, "Thread %u, Base Priority: %5X",
                iNumber, threadEntry.tpBasePri);
      // заносим его в список
      SendMessage ( hListBox, LB_ADDSTRING, 0,
                    ( LPARAM) ( LPCTSTR) buffer);
      // увеличиваем счетчик
      iNumber ++;
     }
     // переходим к поиску следующего потока
     bEnd = Thread32Next ( hSnapshot, &threadEntry);
    }
```

```
   // закрываем дескриптор поиска
   CloseHandle ( hSnapshot);
}
```

В примере вся полученная информация заносится в список, дескриптор которого можно получить, например, с помощью функции GetDlgItem. Кроме перечисления процессов и потоков, имеется возможность непосредственного управления ими: изменение приоритета и принудительное завершение работы.

Каждый процесс или поток имеет определенный уровень привилегий, суть которого заключается в очередности доступа к процессорному времени. Более высокий уровень обрабатывается процессором в первую очередь. Все варианты уровня привилегий для процессов можно свести к четырем: низкий (IDLE_PRIORITY_CLASS), нормальный (NORMAL_PRIORITY_CLASS), высокий (HIGH_PRIORITY_CLASS) и уровень реального времени (REALTIME_PRIORITY_CLASS). По умолчанию любой пользовательский процесс получает нормальный уровень привилегий. Если ваша задача не требует немедленного выполнения, ей следует присвоить низкий уровень, а для важных и быстрых вычислений можно установить высокий уровень. Присваивать процессу уровень реального времени не рекомендуется, поскольку это может привести к сбою всей системы. К тому же, не следует думать, что в Windows можно по-настоящему получить доступ к процессору в реальном времени. Это миф, и любая задача, связанная с реальным временем выполнения (особенно на опасных производствах), должна работать с DOS, UNIX или с аналогичными операционными системами. Чтобы изменить приоритет того или иного процесса, можно использовать функцию SetPriorityClass, одним из аргументов которой является идентификатор процесса. Получить его можно либо с помощью функции GetCurrentProcessId (для текущего процесса), либо перечислением, как было показано раньше. В профессиональных системах Windows (NT/2000/XP/SR3) следует предварительно получить права на доступ к системе. Как это реализуется на практике, показано в листинге 19.16.

Листинг 19.16. Изменение приоритета для процесса

```
#include "stdafx.h"
// пишем функцию для установки приоритета
void SetPriority ( DWORD dwProcessID, DWORD dwPriority)
{
   HANDLE hProcess = 0; // дескриптор открытого процесса
   // открываем процесс
   hProcess = OpenProcess ( PROCESS_SET_INFORMATION, FALSE,
                            dwProcessID);
```

```
   if ( hProcess == NULL) return; // не удалось открыть процесс
   // для Windows NT/2000/XP/SR3 устанавливаем права доступа
   if ( !SetAccess ( hProcess, dwProcessID))
   {
     // если не удалось получить доступ, выходим из функции
     CloseHandle ( hProcess);
     return;
   }
   // устанавливаем приоритет
   DWORD dwResult = SetPriorityClass ( hProcess, dwPriority);
   // если произошла ошибка, выходим из функции
   if ( dwResult == 0)
   {
     MessageBox ( NULL, _T ( "Не удалось изменить уровень приоритета."),
                        _T ( "Отчет"), MB_OK);
     CloseHandle ( hProcess);
     return;
   }
   MessageBox ( NULL, _T ( "Операция успешно выполнена."),
                      _T ( "Отчет"), MB_OK);
   // закрываем процесс
   CloseHandle ( hProcess);
}
// функция для установки прав доступа
bool SetAccess ( HANDLE hProcess, DWORD dwProcessID)
{
   HANDLE hHandleToken = NULL;
   TOKEN_PRIVILEGES newP, curP;
   DWORD dwResult = sizeof ( curP);
   // открываем доступ к текущему потоку
   if ( !OpenThreadToken ( GetCurrentThread (), TOKEN_ADJUST_PRIVILEGES
        | TOKEN_QUERY, false, &hHandleToken))
   {
     // определяем тип ошибки
     if ( GetLastError () != ERROR_NO_TOKEN)
        return false;
     // открываем доступ к текущему процессу
     if ( !OpenProcessToken ( GetCurrentProcess (), TOKEN_QUERY |
           TOKEN_ADJUST_PRIVILEGES, &hHandleToken))
        return false;
   }
   // заполняем информационную структуру доступа
   newP.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
   newP.PrivilegeCount = 1;
```

```
    // определяем локальный идентификатор
    LookupPrivilegeValue ( NULL, SE_DEBUG_NAME,
                           &newP.Privileges[0].Luid);
    // включаем доступ
    if ( !AdjustTokenPrivileges ( hHandleToken, false, &newP,
                                  sizeof ( newP), &curP, &dwResult))
    {
      // если произошла ошибка, выходим из функции
      CloseHandle ( hHandleToken);
      return false;
    }
    // открываем процесс для завершения
    hProcess = OpenProcess ( PROCESS_TERMINATE, false, dwProcessID);
    // восстанавливаем прежний уровень доступа
    AdjustTokenPrivileges ( hHandleToken, false, &curP, sizeof ( curP),
                            NULL, NULL);
    // закрываем дескриптор
    CloseHandle( hHandleToken);
    // если не удалось открыть процесс, возвращаем ошибку
    if ( hProcess == NULL)
       return false;
    // выходим из функции
    return true;
}
// пример работы с функцией SetPriority
DWORD processID = 0L;
// получаем идентификатор текущего процесса
processID = GetCurrentProcessId ();
// устанавливаем высокий уровень приоритета
SetPriority ( processID, HIGH_PRIORITY_CLASS);
```

Для того чтобы принудительно завершить какой-либо процесс, следует воспользоваться функцией `TerminateProcess`. Кроме того, для профессиональных систем потребуется установить права доступа. Пример функции завершения процесса представлен в листинге 19.17.

Листинг 19.17. Завершение работы процесса

```
// пишем функцию завершения работы процесса
void KillProcess ( DWORD dwProcessID, bool bIsNT)
{
   HANDLE hProcess = NULL;
```

```
  // определяем тип системы
  if ( bIsNT) // Windows NT/2000/XP/SR3
  {
    // открываем процесс
    hProcess = OpenProcess ( PROCESS_TERMINATE, false, dwProcessID);
    if ( hProcess == NULL) return;
    // устанавливаем права доступа
    if ( !SetAccess ( hProcess, dwProcessID))
    {
      // если не удалось получить доступ, выходим из функции
      CloseHandle ( hProcess);
      return;
    }
    // пытаемся завершить процесс
    bool bResult = TerminateProcess ( hProcess, ( UINT) -1);
    // если результат равен 0, произошла ошибка
    if ( !bResult)
    {
      CloseHandle ( hProcess);
      return;
    }
    // закрываем дескриптор
    CloseHandle ( hProcess);
  }
  else // Windows 95/98/ME
  {
    // открываем процесс
    hProcess = OpenProcess ( PROCESS_ALL_ACCESS, false, dwProcessID);
    if ( hProcess == NULL) return;
    // пытаемся завершить процесс
    bool bResult = TerminateProcess ( hProcess, ( UINT) -1);
    // если результат равен 0, произошла ошибка
    if ( !bResult)
    {
      CloseHandle ( hProcess);
      return;
    }
    // закрываем дескриптор
    CloseHandle ( hProcess);
  }
}
```

Как видите, завершить процесс довольно просто, однако это не всегда получается. Иногда для лучшего эффекта имеет смысл сначала завершить открытые потоки внутри процесса (если их больше, чем один), а потом закрыть и сам процесс. Для завершения потока применяется функция `TerminateThread`, первый аргумент которой представляет собой дескриптор потока. Его можно получить из функции `OpenThread`. Кроме того, потоки также поддерживают уровень доступа, установить который можно, вызвав функцию `SetThreadPriority`. Общий принцип работы с отдельными потоками мало отличается от управления процессами, поэтому эту тему я оставлю для самостоятельного изучения читателям.

ГЛАВА 20

Работа с файлами

В этой главе мы познакомимся с программированием файлов в операционных системах Windows. Как ни странно, но многие книги практически не затрагивают данную тему, хотя она, по моему мнению, является базовой для любого программиста. Мы рассмотрим три основных темы:

1. Общие принципы работы с файлами.
2. Сжатые файлы.
3. Шифрование файлов.

20.1. Общие принципы

Существуют два основных способа доступа к файлам, каждый из которых имеет свои достоинства и недостатки. Первый заключается в применении стандартных функций Win32 API: `CreateFile`, `ReadFile`, `WriteFile`, `CopyFile`, `CreateDirectory`, `DeleteFile`, `LockFile`, `RemoveDirectory`, `MoveFile`, `SetFilePointer` и `UnlockFile`. Дополнительно существуют также сервисные функции для поиска и получения информации о файлах. Второй способ связан с применением стандартных функций из библиотеки языка С. Он более простой и понятный и к тому же полностью поддерживается всеми операционными системами Windows. Я опишу оба варианта, а вы сами решайте, какой вам больше нравится.

20.1.1. Файловые функции Win32

Функция `CreateFile` универсальна и позволяет создавать и открывать файлы любых типов, а также работать с портами или драйверами. Главным ее недостатком является громоздкая и неудобная структура. Вместе с ней используются функции `ReadFile` (чтение данных из файла) и `WriteFile` (запись

данных в файл). Примеры правильных вызовов этих функций показаны в листинге 20.1.

Листинг 20.1. Использование функций CreateFile, ReadFile и WriteFile

```
// напишем функцию для чтения данных из файла
bool ReadCurrentFile ( char* file, PVOID pOutBuffer)
{
   // проверяем наличие буфера для данных
   if ( !pOutBuffer) return false;
   // объявляем переменные
   DWORD dwFileSize = 0, dwBytesRead = 0;
   // открываем существующий файл для чтения
   HANDLE hFile = CreateFile ( file, GENERIC_READ, 0, NULL,
                           OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибка, выходим из функции
   if ( INVALID_HANDLE_VALUE == hFile) return false;
   // определяем размер файла
   dwFileSize = GetFileSize ( hFile, NULL);
   // устанавливаем указатель на начало файла
   SetFilePointer ( hFile, 0, NULL, FILE_BEGIN);
   // читаем данные из файла
   ReadFile ( hFile, pOutBuffer, dwFileSize, &dwBytesRead, NULL);
   // закрываем файл
   CloseHandle ( hFile);
   // выходим из функции
   return true;
}
// используем функцию ReadCurrentFile для чтения файла с диска
char buffer[24000] = " "; // буфер для считываемых данных
// читаем файл
ReadAllFile ( "C:\\autoexec.bat", buffer);
// пишем функцию для создания нового файла
bool CreateNewFile ( char* file)
{
   // создаем новый файл
   HANDLE hFile = CreateFile ( file, GENERIC_READ, 0, NULL,
             CREATE_ALWAYS | CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибка, выходим из функции
   if ( INVALID_HANDLE_VALUE == hFile) return false;
   // закрываем файл
   CloseHandle ( hFile);
}
```

```
// используем функцию CreateNewFile для создания пустых файлов
CreateNewFile ( "C:\\MyListing.txt");
CreateNewFile ( "D:\\MyData.dat");
// пишем функцию для записи данных в файл
bool WriteCurrentFile ( char* file, PVOID pInBuffer, DWORD dwSizeData)
{
   // проверяем наличие буфера с данными
   if ( !pInBuffer) return false;
   // объявляем переменные
   DWORD dwBytesWrite = 0;
   // открываем существующий файл для чтения
   HANDLE hFile = CreateFile ( file, GENERIC_WRITE, 0, NULL,
                               CREATE_NEW, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибка, выходим из функции
   if ( INVALID_HANDLE_VALUE == hFile) return false;
   // устанавливаем указатель на начало файла
   SetFilePointer ( hFile, 0, NULL, FILE_BEGIN);
   // пишем данные в файл
   WriteFile ( hFile, pInBuffer, dwSizeData, &dwBytesWrite, NULL);
   // закрываем файл
   CloseHandle ( hFile);
   // выходим из функции
   return true;
}
// пример использования функции WriteCurrentFile
char buffer[5000] = " "; // буфер с данными
// создаем новый файл и записываем в него данные
WriteCurrentFile ( "C:\\MyLog.txt", buffer, sizeof ( buffer));
```

Примеры, рассмотренные выше, упрощены для лучшего понимания работы файловых функций. Например, для чтения файла сначала следует определить его размер, выделить достаточный буфер (или использовать цикл для чтения), а только потом производить считывание данных.

Функции `LockFile` и `UnlockFile` позволяют блокировать доступ к определенной области открытого файла, чтобы к ней нельзя было обратиться (прочитать или записать) из других процессов. Эти функции имеет смысл применять, если нужно защитить файл от записи или чтения из других программ на время обработки общедоступных данных (например, записи в файл базы данных). Пример использования функций блокировки представлен в листинге 20.2.

Листинг 20.2. Блокировка доступа к файлу

```
DWORD dwPosition = 0; // текущая позиция чтения
DWORD dwBytesRead = 0; // число прочитанных байтов
```

```
char buffer[9600] = ""; // буфер для считываемых данных
// открываем существующий файл для чтения
HANDLE hFile = CreateFile ( "c:\\config.sys", GENERIC_READ, 0, NULL,
                            OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
// устанавливаем указатель на начало файла
dwPosition = SetFilePointer ( hFile, 0, NULL, FILE_BEGIN);
// блокируем часть файла перед чтением
LockFile ( hFile, dwPosition, 0, dwPosition + 9600, 0);
// читаем данные из файла
ReadFile ( hFile, buffer, 9600, &dwBytesRead, NULL);
// разблокируем данные
UnlockFile ( hFile, dwPos, 0, dwPosition + 9600, 0);
// закрываем файл
CloseHandle ( hFile);
```

Функция `SetFilePointer` предназначена для управления внутренним указателем позиции в открытом файле. Данная функция незаменима при произвольном доступе к файлу и позволяет определить размер файла. В листинге 20.3 представлены два примера ее использования.

Листинг 20.3. Работа с функцией `SetFilePointer`

```
// пишем функцию для определения размера файла
DWORD GetSize ( char* file)
{
   DWORD dwFileSize = 0; // размер файла
// открываем существующий файл для чтения
HANDLE hFile = CreateFile ( file, GENERIC_READ, 0, NULL,
                            OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
// устанавливаем указатель на начало файла
SetFilePointer ( hFile, 0, NULL, FILE_BEGIN);
// устанавливаем указатель на конец файла
dwFileSize = SetFilePointer ( hFile, 0, NULL, FILE_END);
// закрываем файл
CloseHandle ( hFile);
// возвращаем полный размер файла в байтах
return dwFileSize;
}
// пишем функцию для приведения к нулевому размеру указанного файла
void SetFileZero ( char* file)
{
   // открываем существующий файл для записи
   HANDLE hFile = CreateFile ( file, GENERIC_WRITE, 0, NULL,
                               OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
    // устанавливаем указатель на начало файла
    SetFilePointer ( hFile, 0, NULL, FILE_BEGIN);
    // записываем признак конца файла
    SetEndOfFile ( hFile);
    // закрываем файл
    CloseHandle ( hFile);
}
```

Функции CopyFile, DeleteFile и MoveFile позволяют соответственно копировать, удалять или перемещать указанный файл. С помощью функции CopyFile можно не только выполнять копирование с одного диска на другой, но и переименовывать любой доступный файл. Первые два аргумента функции определяют имена и пути исходного и конечного файлов. Последний аргумент устанавливает действие при обнаружении аналогичного файла (если он равен FALSE, то существующий файл будет перезаписан). Функция MoveFile при необходимости также выполняет переименование перемещаемого файла. Кроме того, она поддерживает перемещение целых каталогов со всеми подкаталогами и файлами. Однако все перемещения могут быть осуществлены лишь в пределах одного логического диска. Чтобы переместить файл между дисками, сначала следует скопировать его, а затем удалить исходную копию.

Для создания и удаления каталогов применяются функции CreateDirectory и RemoveDirectory. Важно помнить, что удалить можно только пустой каталог, не имеющий вложенных каталогов и файлов.

Примеры использования рассмотренных функций приведены в листинге 20.4.

Листинг 20.4. Работа с файловыми функциями

```
// копировать файл без переименования
CopyFile ( "C:\\Windows\\Win.ini", "D:\\Win.ini", false);
// скопировать и переименовать файл
CopyFile ( "C:\\Windows\\Win.ini", "D:\\Win.log", false);
// скопировать файл, если не существует аналога
CopyFile ( "C:\\Windows\\Calc.exe", "D:\\Calc.exe", true);
// переместить файл из одного каталога в другой
MoveFile ( "F:\\MyMusic.wav", "F:\\Music\\MyMusic.wav");
// переместить файл из одного каталога в другой с переименованием
MoveFile ( "F:\\MyMusic.wav", "F:\\Music\\Melody.wav");
// переместить каталог без переименования
MoveFile ( "F:\\Temp", "F:\\Music\\Temp");
// переместить каталог с переименованием
MoveFile ( "F:\\Temp", "F:\\Music\\Temp_2");
```

```
// удалить файл
DeleteFile ( "C:\\ErrorReport.log");
// создать новый каталог
CreateDirectory ( "C:\\Temp", NULL);
// удалить каталог
RemoveDirectory ( "C:\\Temp");
```

Пример удаления каталога с файлами и подкаталогами рассмотрен в листинге 20.5.

Листинг 20.5. Удаление каталога с файлами и подкаталогами

```
// пишем функцию удаления каталога
void DelFolder ( char* path)
{
   HANDLE hFind = NULL; // дескриптор поиска подкаталогов и файлов
   WIN32_FIND_DATA ff; // структура поиска
   char buffer[MAX_PATH] = ""; // временный буфер
   char* pName = NULL; // указатель на имя файла
   // обнуляем структуру
   ZeroMemory ( &ff, sizeof ( WIN32_FIND_DATA));
   // создаем маску для поиска файлов
   sprintf ( buffer, "%s\\*.*", path);
   // ищем первый файловый объект
   hFind = FindFirstFile ( buffer, &ff);
   // если ошибка, выходим из функции
   if ( hFind == INVALID_HANDLE_VALUE)
        return false;
   // создаем цикл
   do
   {
     // получаем указатель на имя найденного файлового объекта
     pName = ff.cFileName;
     // пропускаем лишнее
     if ( pName[1] == 0 && *pName == '.')
          continue;
     else if ( pName[2] == 0 && *pName == '.' && pName[1] == '.')
          continue;
     // записываем имя найденного объекта и организуем путь
     sprintf ( buffer, "%s\\%s", path, ff.cFileName);
     // если объект защищен от записи, снимаем атрибут
       if ( ff.dwFileAttributes & FILE_ATTRIBUTE_READONLY)
            SetFileAttributes ( buffer,
                       ff.dwFileAttributes & ~FILE_ATTRIBUTE_READONLY);
```

```
      // если найденный объект является каталогом
      if ( ff.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
      {
        // рекурсия
        DelFolder ( buffer);
      }
      else
      {
        // удаляем файл
        DeleteFile ( buffer);
      }
    }
    // ищем следующий файл
    while ( FindNextFile ( hFind, &ff));
    // закрываем дескриптор поиска
    FindClose ( hFind);
    // удаляем корневую папку
    if ( ff.dwFileAttributes & FILE_ATTRIBUTE_READONLY)
      // если необходимо, снимаем атрибут
      SetFileAttributes ( path,
                   ff.dwFileAttributes & ~FILE_ATTRIBUTE_READONLY);
    // удаляем корневую папку
    RemoveDirectory ( path);
}
```

Приведенный выше код будет без проблем работать в Windows 9*x*/ME, однако в профессиональных системах необходимо дополнительно получить права доступа для удаления каталога с диска. Пример кода, демонстрирующий регистрацию прав на удаление каталога, представлен в листинге 20.6.

Листинг 20.6. Получение прав на удаление каталога в Windows NT/2000/XP/SR3

```
// пишем функцию инициализации для установки прав доступа к объекту
bool InitAccess ( char* object)
{
    // дескриптор доступа для текущего процесса
    HANDLE hHandleToken = NULL;
    int file_Object[255];
    DWORD handle = 0;
    // идентификация пользователя объекта
    TOKEN_USER *user = ( TOKEN_USER*) file_Object;
    // дескриптор безопасности
    SECURITY_DESCRIPTOR sec_handle;
```

```
    // информационная структура дескриптора безопасности
    SECURITY_INFORMATION info_user = OWNER_SECURITY_INFORMATION;
    // инициализируем новый дескриптор безопасности
    InitializeSecurityDescriptor ( &sec_handle,
                                   SECURITY_DESCRIPTOR_REVISION);
    // открываем дескриптор доступа для текущего процесса
    if ( OpenProcessToken ( GetCurrentProcess (), TOKEN_READ |
                            TOKEN_ADJUST_PRIVILEGES, & hHandleToken))
    {
      // устанавливаем уровень доступа
      if ( SetPrivilege ( hHandleToken, SE_TAKE_OWNERSHIP_NAME, false))
      {
       // получаем необходимую информацию о дескрипторе доступа
       if ( GetTokenInformation ( hHandleToken, TokenUser, user,
                                  sizeof ( file_Object), &handle))
       {
        // заменяем дескриптор безопасности
        if ( SetSecurityDescriptorOwner ( &sec_handle,
                                          user->User.Sid, false))
        {
          // устанавливаем защиту для файлового объекта
          if ( SetFileSecurity ( object, info_user, &sec_handle))
               return true; // если нет ошибок
        }
       }
      }
    }
    // отключаем права доступа
    LoadPrivilege ( hHandleToken, SE_TAKE_OWNERSHIP_NAME, true);
    // выходим из функции с ошибкой
    return false;
}
// пишем функцию доступа к файловому объекту
bool SetAccessFile ( const char* object, const char* pszAccess,
                     DWORD dwMask)
{
    // определяем переменные
    SECURITY_DESCRIPTOR sec_handle;
    SECURITY_INFORMATION info_sec = DACL_SECURITY_INFORMATION;
    int tmp[512];
    char domain_name[512];
    SID_NAME_USE sid_name;
    PSID sid = ( PSID*) tmp;
```

```
   ACL *list;
   DWORD domain_size = 0L, sid_size= 0L, list_size = 0L;
   // инициализируем новый дескриптор безопасности
   InitializeSecurityDescriptor ( &sec_handle,
                                 SECURITY_DESCRIPTOR_REVISION);
   // получаем идентификатор безопасности SID
   if ( LookupAccountName ( NULL, pszAccess, sid, &sid_size, domain_name,
                             &domain_size, &sid_name))
   {
     // инициализируем список доступа
     list_size = sizeof ( ACL) + sizeof ( ACCESS_ALLOWED_ACE) +
                 GetLengthSid ( tmp) - sizeof ( DWORD);
     // выделяем память из кучи
     list = ( ACL *) HeapAlloc ( GetProcessHeap (), HEAP_ZERO_MEMORY,
                                 list_size);
     // создаем новый список доступа
     InitializeAcl ( list, list_size, ACL_REVISION);
     // открываем доступ для указанного идентификатора безопасности
     if ( AddAccessAllowedAce ( list, ACL_REVISION, dwMask, sid))
     {
       // добавляем в список доступа необходимую информацию
       if ( SetSecurityDescriptorDacl ( &sec_handle, true, list,
                                        false))
       {
         // устанавливаем защиту для файлового объекта
         if ( SetFileSecurity ( object, info_sec, &sec_handle))
              return true; // если нет ошибок
       }
     }
   }
   // освобождаем память
   if ( list != NULL)
       HeapFree ( GetProcessHeap (), 0, ( LPVOID) list);
// выходим из функции с ошибкой
return false;
}
// пишем функцию для установки уровня привилегий
void LoadPrivilege ( HANDLE hHandleToken, const char* pszData,
                     bool bSet)
{
   LUID luID; // уникальный идентификатор
   TOKEN_PRIVILEGES token_priv; // информация о доступе
```

```
    // если требуется определить параметры доступа
    if ( !bSet && pszData != NULL)
    {
      // получаем идентификатор
      if ( LookupPrivilegeValue ( NULL, pszData, &luID))
      {
       // определяем параметры доступа
       token_priv.Privileges[0].Luid = luID;
       token_priv.PrivilegeCount = 1;
       token_priv.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
      }
    }
    // разрешаем или запрещаем привилегии
    AdjustTokenPrivileges ( hHandleToken, bSet,& token_priv,
                sizeof ( TOKEN_PRIVILEGES), ( PTOKEN_PRIVILEGES) NULL,
                ( PDWORD) NULL);
}
// добавляем функции установки прав доступа в основной код
// . . .
// если найденный объект является каталогом
if ( ff.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
{
    // установка доступа
    if ( InitAccess ( buffer))
    {
      // получаем доступ к файловому объекту
      SetAccessFile ( buffer, "set_enable_access", GENERIC_ALL);
    }
    // рекурсия
    DelFolder ( buffer);
}
else
{
    // установка доступа
    if ( InitAccess ( buffer))
    {
      // получаем доступ к файловому объекту
      SetAccessFile ( buffer, "set_enable_access", GENERIC_ALL);
    }
    // удаляем файл
    DeleteFile ( buffer);
}
// . . .
```

Кроме основных файловых функций, имеется набор дополнительных функций, используемых преимущественно для получения и установки различных параметров файловой системы. К ним относятся функции поиска (`FindFirstFile`, `FindNextFile`, `FindClose`), получения и установки атрибутов файла (`SetFileAttributes` и `GetFileAttributes`), сброса кэша на диск (`FlushFileBuffers`), получения и установки текущего каталога (`GetCurrentDirectory` и `SetCurrentDirectory`), определения свободного места на диске (`GetDiskFreeSpaceEx`), получения типа диска (`GetDriveType`) и определения размера файла (`GetFileSize` и `GetFileSizeEx`).

С функциями поиска мы уже познакомились ранее. Добавлю только, что они позволяют найти любые типы файлов или каталогов на указанном логическом диске. Как правило, для поиска вложенных файлов применяется метод рекурсии (вызов функции самой себя), что может привести, в частности, к переполнению стека. Чтобы исключить такую проблему, рекомендую устанавливать ограничительный счетчик на количество рекурсивных вызовов функции, а также создавать саму функцию поиска как статическую. В листинге 20.7 показан пример кода, выполняющий поиск всех файлов с расширением txt на диске С:.

Листинг 20.7. Использование функций поиска файлов

```
// напишем функцию поиска текстовых файлов и добавления их в список
void Find_File_TXT ( HWND hListBox, char* path)
{
   // определяем необходимые переменные
   HANDLE hFind = NULL;
   WIN32_FIND_DATA ff;
   TCHAR buf[MAX_PATH] = "";
   // обнуляем структуру поиска файловых объектов
   ZeroMemory ( &ff, sizeof ( WIN32_FIND_DATA));
   // определяем указатель на путь
   char *ptr = path;
   while ( *ptr) ptr++;
   // добавляем маску поиска
   strcpy ( ptr, "\\*.*");
   // начинаем поиск первого объекта
   hFind = FindFirstFile ( path, &ff);
   // если ошибка, выходим из функции
   if ( hFind != INVALID_HANDLE_VALUE)
   {
     // проверяем все элементы
     do
     {
```

```
      // отсекаем лишнее
      if ( ff.cFileName[0] != '.' ||
         ( ff.cFileName[1] != '.' && ff.cFileName[1]))
      {
       // сохраняем указатель на путь для обработки вложенных папок
       strcpy ( ptr + 1, ff.cFileName); ′
       // если найденный объект является папкой
       if ( ff.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
       {
         // вызываем свою функцию повторно и контролируем ошибки
         try
         {
           Find_File_TXT ( hListBox, path); // рекурсия
         }
         catch ( . . .)
         {
           // переходим к следующей итерации цикла
           continue;
         }
       }
       else // если найден файл
       {
         // копируем найденный файл в буфер
         strcpy ( buf, ff.cFileName);
         // переводим в нижний регистр
         strlwr ( buf);
         // ищем расширение в имени файла
         char* ext = strrchr ( buf, '.');
         // если расширения нет или оно не равно "txt", ищем дальше
         if ( ext != NULL && stricmp (ext, ".txt") == 0)
         {
           // добавляем имя файла в стандартный список
           SendMessage ( hListBox, LB_ADDSTRING, 0,
                         ( LPARAM) ( LPCTSTR) buf);
         }
       }
      }
    // ищем следующий файловый объект
    } while ( FindNextFile ( hFind, &ff));
    // закрываем дескриптор поиска
    FindClose ( hFind);
    hFind = NULL;
  }
}
```

```
// пример вызова функции Find_File_TXT
// получаем дескриптор окна для списка
HWND hListBox = GetDlgItem ( hDlg, IDC_MYLISTBOX);
char path[MAX_PATH] = "";
// указываем букву диска для поиска текстовых файлов
strcpy ( path, "C:");
// вызываем функцию поиска
Find_File_TXT ( hListBox, path);
```

Если вам требуется найти файл (файлы) для операции доступа (чтение, запись) или удаления, сначала следует снять атрибут FILE_ATTRIBUTE_READONLY (только для чтения) и только потом выполнять какие-либо действия.

Для установки или получения текущих атрибутов файла применяются две функции SetFileAttributes и GetFileAttributes. Объекты файловой системы в Windows могут иметь следующие атрибуты:

- FILE_ATTRIBUTE_ARCHIVE — архивный файл или папка, доступный для удаления;
- FILE_ATTRIBUTE_NORMAL — атрибуты отсутствуют;
- FILE_ATTRIBUTE_HIDDEN — скрытый файл или папка;
- FILE_ATTRIBUTE_READONLY — файл или папка доступны только для чтения;
- FILE_ATTRIBUTE_SYSTEM — системный файл или папка;
- FILE_ATTRIBUTE_DIRECTORY — файловый объект является каталогом.

Существуют еще несколько атрибутов, но они используются крайне редко. Значение FILE_ATTRIBUTE_NORMAL отменяется комбинацией любых других атрибутов. В листинге 20.8 приводятся примеры установки и получения атрибутов для файловых объектов.

Листинг 20.8. Установка и получение атрибутов

```
// проверяем различные атрибуты для заданного файла
DWORD dwAttrib = GetFileAttributes ( FileName);
// если только для чтения
if ( dwAttrib & FILE_ATTRIBUTE_READONLY)
{
   // сбрасываем атрибут
   SetFileAttributes ( FileName, dwAttrib ^ FILE_ATTRIBUTE_READONLY);
}
// если не системный
if ( dwAttrib & FILE_ATTRIBUTE_SYSTEM) != FILE_ATTRIBUTE_SYSTEM)
```

```
{
   // устанавливаем атрибут
   SetFileAttributes ( FileName, dwAttrib | FILE_ATTRIBUTE_SYSTEM);
}
// проверяем наличие папки
if ( dwAttrib & FILE_ATTRIBUTE_DIRECTORY) == FILE_ATTRIBUTE_DIRECTORY)
{
   // выполняем какие-либо действия
}
```

Функция `FlushFileBuffers` дает возможность немедленно записать все буферизированные данные для открытого файла на диск. Поскольку операционная система хранит данные в промежуточном буфере (для ускорения работы) и записывает их на диск в удобное для себя время (например, во время простоя), пользователь рискует потерять важные данные при внезапном отключении питания или другом аппаратно-программном сбое. Вызов функции `FlushFileBuffers` после записи всех данных в файл гарантирует их сохранение на диск. Данная функция имеет всего один аргумент (дескриптор открытого файла) и проста в использовании.

Для получения и установки текущего каталога (в текущем процессе) применяются соответственно функции `GetCurrentDirectory` и `SetCurrentDirectory`. Их удобно вызывать перед работой с файлами, расположенными в текущей папке приложения. Первая функция имеет два аргумента: размер и указатель на выделенный буфер. Вторая просто определяет строковое значение для установки текущего каталога. Примеры получения и установки текущего каталога показаны в листинге 20.9.

Листинг 20.9. Получение и установка текущего каталога

```
// получаем текущий путь для нашей программы
TCHAR szCurrentPath[MAX_PATH];
if ( GetCurrentDirectory ( sizeof ( szPath) / sizeof ( TCHAR),
                              szPath))
{
   // выводим сообщение
   MessageBox ( NULL, szPath, "Текущий путь", MB_OK);
}
// устанавливаем новый путь для нашей программы
char path[MAX_PATH];
if ( GetCurrentDirectory ( sizeof ( path), path))
{
   // новый путь
  SetCurrentDirectory ( path);
}
```

```
// открываем файл данных из текущей папки программы
if ( GetCurrentDirectory ( sizeof ( path), path))
{
   // добавляем имя открываемого файла
   strcat ( path, "\\Readmy.txt");
   // открываем и обрабатываем файл
   HANDLE hFile = CreateFile ( path, GENERIC_READ, 0, NULL,
                               OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибок нет
   if ( INVALID_HANDLE_VALUE != hFile)
   {
     // выполняем какие-либо операции над файлом
     // . . .
     // закрываем файл
     CloseHandle ( hFile);
   }
}
```

Следующая функция, которую мы рассмотрим, связана с определением свободного места на логическом диске и называется GetDiskFreeSpaceEx. Существует более старая версия данной функции, поддерживающая диски размером не более 2 Гбайт, но она практически не применяется. Функция GetDiskFreeSpaceEx имеет четыре аргумента. Первый аргумент задает имя диска. Если установить его в NULL, будет возвращена информация для текущего диска. Второй аргумент представляет собой указатель на переменную, содержащую общее число свободного места на диске в байтах для вызывающего потока (доступного пользователю). Третий указывает на общий размер диска. Последний аргумент возвращает общее число свободного места на диске в байтах. Пример работы с данной функцией представлен в листинге 20.10.

Листинг 20.10. Определение свободного места на диске

```
void ShowDiskSize ()
{
  // напишем функцию, определяющую размер диска
  char* disk = "D:\\"; // задаем букву диска
  char buf[100]; // буфер для форматирования данных
  // переменные для хранения результата
  unsigned __int64 ui64FreeBytesUser, ui64TotalBytes,
                                      ui64TotalFreeBytes;
  // вызываем функцию
  if ( GetDiskFreeSpaceEx ( disk, ( PULARGE_INTEGER)&ui64FreeBytesUser,
                                  ( PULARGE_INTEGER) &ui64TotalBytes,
                                ( PULARGE_INTEGER) &ui64TotalFreeBytes))
```

```
  {
    // форматируем полученный результат
    sprintf ( buf, "Total: %I64u MB\n Free: %I64u MB\n \
              Free User: %I64u MB", ui64TotalBytes / ( 1024 * 1024),
                                    ui64TotalFreeBytes / ( 1024 * 1024),
                                     ui64FreeBytesUser / ( 1024 * 1024));
   // выводим на экран сообщение
   MessageBox ( NULL, buf, "Report", MB_OK);
  }
}
```

Функция GetDriveType позволяет определить тип подключенного дисковода. Поддерживаются следующие типы дисков: жесткий, гибкий, сетевой, CD-ROM, виртуальный (созданный в памяти). Функция принимает один аргумент, определяющий букву логического диска. Демонстрационный пример работы с этой функцией показан в листинге 20.11.

Листинг 20.11. Определение типа диска

```
// напишем функцию, позволяющую определить все типы дисков, установленных
// на компьютере, и записать эту информацию в раскрывающийся список
bool LoadDrives ( HWND hComboBox)
{
   DWORD dwDrive = 0;
   unsigned int nDriveType = 0;
   int nPos = 0;
   char buf[20] = "";
   int ch = 0;
   // получаем все логические диски, имеющиеся в системе
   try
   {
     dwDrive = GetLogicalDrives ();
     // перечисляем найденные диски
     while ( dwDrive)
     {
       // получаем букву диска и добавляем атрибуты
       if ( dwDrive & 1)
       {
         ch = 0x41 + nPos;
         lstrcpy ( buf, ( LPCTSTR) &ch);
         lstrcat ( buf, ":\\");
         // определяем тип диска
         nDriveType = GetDriveType ( buf);
```

```
        lstrcpyn ( buf, buf, 3);
        switch ( nDriveType)
        {
        case DRIVE_REMOVABLE:
          lstrcat ( buf, " ( гибкий диск)");
          break;
        case DRIVE_FIXED:
          lstrcat ( buf, " ( жесткий диск)");
          break;
        case DRIVE_CDROM:
          lstrcat ( buf, " ( CD-ROM)");
          break;
        case DRIVE_REMOTE:
          lstrcat ( buf, " ( сетевой диск)");
          break;
        case DRIVE_RAMDISK:
          lstrcat ( buf, " ( виртуальный диск)");
          break;
        }
      }
      // добавляем диск в список
      SendMessage ( hComboBox, CB_ADDSTRING, 0, ( LPARAM) buf);
      // переходим к следующему диску
      dwDrive >>= 1;
      nPos++;
    }
  }
  catch ( . . .)
  {
    return false; // ошибка
  }
  return true; // успешное выполнение
}
```

Функции GetFileSize и GetFileSizeEx позволяют определить размер файла в байтах, открытого с использованием одного из флагов: GENERIC_WRITE или GENERIC_READ. Первая функция имеет два аргумента — дескриптор открытого файла и указатель на старшее 16-разрядное значение размера. Второй аргумент применяется для измерения больших файлов, иначе он должен быть установлен в NULL. При работе с большими файлами можно вызывать функцию GetFileSizeEx (только для Windows NT/2000/XP/SR3). Она также имеет два аргумента, но поддерживает очень большие файлы.

В VC++ версии 6.0 определение функции GetFileSizeEx может отсутствовать, что приведет к ошибке компиляции. Данную проблему можно решить

одним из трех способов: вызвать функцию динамически, обновить файл Winbase.h или использовать оболочку VC++ .NET.

Кроме перечисленных выше, существует еще одна интересная и полезная функция для определения реальных размеров сжатого файла, которая называется `GetCompressedFileSize`. Ее первый аргумент указывает на имя файла, а второй — на переменную, в которую будет помещено старшее 16-разрядное значение результата. К сожалению, эта функция предназначена только для профессиональных операционных систем (Windows NT/2000/XP/SR3). Примеры работы с рассмотренными функциями показаны в листинге 20.12.

Листинг 20.12. Определение размера файла

```
// использование функции GetFileSize
// получаем размер стандартного файла
DWORD GetSize ( const char* file)
{
   DWORD dwFileSize = 0;
   // открываем существующий файл для чтения
   HANDLE hFile = CreateFile ( file, GENERIC_READ, 0, NULL,
                              OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибка, выходим из функции
   if ( INVALID_HANDLE_VALUE == hFile) return 0L;
   // определяем размер файла
   dwFileSize = GetFileSize ( hFile, NULL);
   // закрываем дескриптор файла
   CloseHandle ( hFile);
   // возвращаем результат
   return dwFileSize;
}
// получаем размер большого файла
DWORD GetLargeSize ( const char* file)
{
   DWORD dwLoSize = 0, dwHiSize = 0;
   // открываем существующий файл для чтения
   HANDLE hFile = CreateFile ( file, GENERIC_READ, 0, NULL,
                              OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибка, выходим из функции
   if ( INVALID_HANDLE_VALUE == hFile) return 0L;
   // определяем размер файла
   dwLoSize = GetFileSize ( hFile, &dwHiSize);
   // закрываем дескриптор файла
   CloseHandle ( hFile);
```

```
   // проверяем ошибки
   if ( ( NO_ERROR != GetLastError ()) && ( dwLoSize == -1))
       return 0L; // ошибка
   // возвращаем результат
   return ( MAKELONG ( dwLoSize, dwHiSize));
}
// использование функции GetFileSizeEx
// получаем размер любого поддерживаемого в Windows файла
unsigned __int64 GetAnyFileSize ( const char* file)
{
   unsigned __int64 ui64TotalSize = 0;
   // открываем существующий файл для чтения
   HANDLE hFile = CreateFile ( file, GENERIC_READ, 0, NULL,
                               OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
   // если ошибка, выходим из функции
   if ( INVALID_HANDLE_VALUE == hFile) return 0L;
   // определяем размер файла
   bool bResult = GetFileSizeEx ( hFile, &ui64TotalSize);
   // закрываем дескриптор файла
   CloseHandle ( hFile);
   // проверяем ошибки
   if ( !bResult) return 0L; // ошибка
   // возвращаем результат
   return ui64TotalSize;
}
// использование функции GetCompressedFileSize
// получаем реальный размер сжатого файла
DWORD GetSizeArchiveFile ( const char* file)
{
   DWORD dwFileSize = 0;
   // получаем размер файла
   dwFileSize = GetCompressedFileSize ( file, NULL);
   // проверяем ошибки
   if ( dwFileSize == -1) return 0L;
   // возвращаем результат
   return dwFileSize;
}
```

Кроме стандартных средств Win32 API, для поддержки файловых операций в реальных программах применяются потоковые функции языка С, полностью совместимые с интерфейсом Win32 API.

20.1.2. Файловые функции языка C

Наиболее часто используются следующие функции: fopen (открытие файла), _fsopen (открытие общего файла), fclose (закрытие файла), fread (чтение данных), fwrite (запись данных), fseek (управление указателем позиции в файле), fflush (сброс всех буферов на диск), fgetc (чтение одного символа из потока), fputc (запись символа в поток), fgets (чтение строки символов из потока), fputs (запись строки символов в поток), fprintf (запись отформатированных данных в поток), fscanf (чтение отформатированных данных из потока), ftell (получение текущей позиции в файле). Кроме перечисленных, имеются и другие функции, о которых можно прочитать в любой книге по программированию в C/C++.

Для открытия или создания нового файла применяется функция fopen. Она имеет два аргумента: имя файла и режим доступа, а в случае успеха возвращает указатель на открытый файл (FILE*). Примеры работы с ней показаны в листинге 20.13.

Листинг 20.13. Использование функции fopen

```
#include <stdio.h>
// дескриптор файла
FILE* file = NULL;
// открываем как двоичный файл для чтения
if ( ( file  = fopen ( "c:\\autoexec.bat", "rb")) == NULL)
{
   // ошибка, не удалось открыть указанный файл
}
// открываем как двоичный файл для записи в конец
if ( ( file  = fopen ( "c:\\autoexec.bat", "ab")) == NULL)
{
   // ошибка, не удалось открыть указанный файл
}
// создаем как двоичный файл для записи
if ( ( file  = fopen ( "c:\\MyLog.log", "wb")) == NULL)
{
   // ошибка, не удалось создать указанный файл
}
// открываем как двоичный файл для чтения и записи
if ( ( file  = fopen ( "c:\\MyProg.ini ", "rb+")) == NULL)
{
   // ошибка, не удалось открыть указанный файл
}
```

```
// создаем двоичный файл для чтения и записи
if ( ( file  = fopen ( "c:\\MyProg.ini ", "wb+")) == NULL)
{
   // ошибка, не удалось создать указанный файл
}
// открываем как текстовый файл для чтения
if ( ( file  = fopen ( "c:\\autoexec.bat", "r")) == NULL)
{
   // ошибка, не удалось открыть указанный файл
}
// создаем текстовый файл для записи
if ( ( file  = fopen ( "c:\\MyLog.log", "w")) == NULL)
{
   // ошибка, не удалось создать указанный файл
}
```

Функция _fsopen аналогична предыдущей, за исключением одной важной особенности — она позволяет обращаться к файлам, уже используемым другими программами. Как правило, данная функция позволяет решить некоторые специфические задачи. В листинге 20.14 показан пример работы с функцией _fsopen.

Листинг 20.14. Использование функции _fsopen

```
#include <share.h>
FILE* file = NULL; // дескриптор файла
char buf[101]; // буфер для данных
// открываем наш выполняемый файл программы (сами себя)
// для чтения и записи
if ( ( file  = _fsopen ( ExeName, "rb", _SH_DENYNO)) == NULL)
{
   // ошибка, не удалось открыть указанный файл
}
// установим указатель на начало блока каких-либо данных,
// расположенных в конце нашего выполняемого модуля
fseek ( file, -100, SEEK_END);
// читаем данные в буфер
fread ( &buf, 1, sizeof ( buf), file);
// закрываем файл
fclose ( file);
// обрабатываем полученные данные
//  . . .
```

В нашем примере мы открыли собственный выполняемый модуль (файл с расширением exe) и прочитали ранее сохраненные в нем данные. Такую методику работы с файлами можно применять для различных трюков, извлекая необходимую информацию из "самого себя". При компиляции программы не забудьте добавить файл определений share.h в начало кода.

Функции `fread` и `fwrite` предназначены соответственно для чтения и записи данных в открытый файл. Обе они имеют по четыре аргумента: указатель на буфер данных, размер блока данных, число блоков и указатель на дескриптор открытого файла. Как ими пользоваться, показано в листинге 20.15.

Листинг 20.15. Операции чтения и записи в файл

```
#include <stdio.h>
// пишем функцию для чтения из одного двоичного файла в другой
void ReadFileData ( FILE* src, FILE* dest, char* pBuffer, DWORD dwSize)
{
   DWORD dwCurrentRead = 0, dwReadBytes = 0;
   // запускаем цикл обработки
   while ( dwCurrentRead < dwSize)
   {
     // читаем блок данных из исходного файла
     dwReadBytes = fread ( pBuffer, 1, 2048, src);
     // записываем прочитанные данные в конечный файл
     fwrite ( pBuffer, dwReadBytes, 1, dest);
     // увеличиваем счетчик прочитанных байтов
     dwCurrentRead += dwReadBytes;
   }
}
// пишем функцию для определения размера открытого файла
long GetSizeAnyFile ( FILE* file)
{
   long current = 0, len = 0;
   // сохраняем текущую позицию в файле
   current = ftell ( file);
   // переводим указатель в конец файла
   fseek ( file, 0L, SEEK_END);
   // получаем текущую позицию в файле
   len = ftell ( file);
   // возвращаем указатель на прежнюю позицию
   fseek ( file, current, SEEK_SET);
   // возвращаем полный размер файла в байтах
   return len;
}
```

```
// пишем собственно функцию копирования файла
bool CopyAnyFile ( const char* src_file, const char* dest_file)
{
   // указатель на буфер данных
   char* pBuffer = NULL;
   // размер исходного файла
   DWORD dwFileSize = 0;
   // дескрипторы исходного и конечного файлов
   FILE* src = NULL;
   FILE* dest = NULL;
   // инициализируем буфер для данных
   pBuffer = new char [2048 + 1];
   if ( pBuffer == NULL) return false; // нет свободной памяти
   // создаем и открываем конечный файл
   if ( ( dest  = fopen ( dest_file, "wb")) == NULL)
   {
     // если ошибка, освобождаем память
     delete [] pBuffer;
     pBuffer = NULL;
     return false; // возвращаем ошибку
   }
   // устанавливаем курсор на начало файла
   fseek ( dest, 0L, SEEK_SET);
   // открываем исходный файл для чтения
   if ( ( src  = fopen ( src_file, "rb")) == NULL)
   {
     // если ошибка, освобождаем память
     delete [] pBuffer;
     pBuffer = NULL;
     // и закрываем конечный файл
     fclose ( dest);
     return false; // возвращаем ошибку
   }
   // получаем размер исходного файла
   dwFileSize = GetSizeAnyFile ( src);
   // копируем данные из одного файла в другой
   ReadFileData ( src, dest, pBuffer, dwFileSize);
   // освобождаем память
   delete [] pBuffer;
   pBuffer = NULL;
   // закрываем файлы
   fclose ( dest);
   fclose ( src);
```

```
    // функция успешно выполнена
    return true;
}
```

Кроме операций чтения и записи, в рассмотренном примере мы использовали функции ftell и fseek, чтобы определить полный размер исходного файла. Первая из них возвращает текущую позицию (в байтах) в открытом файле, а вторая устанавливает новое значение. Нехитрая комбинация этих функций позволила вычислить размер файла в байтах.

Функция fflush используется, если необходимо немедленно сохранить данные на диск. Она аналогична рассмотренной ранее функции FlushFileBuffers, только принимает в качестве аргумента указатель на дескриптор открытого файла (FILE*).

Для чтения или записи отдельных символов применяют функции fgetc и fputc. Их наиболее ценные качества проявляются при обработке текстовых файлов. Например, чтобы прочитать текстовый файл в стандартный список (List Box), используйте код, показанный в листинге 20.16. Предположим, что в файле хранится символический плейлист (имя и путь) для музыкального проигрывателя.

Листинг 20.16. Использование функции fgetc

```
#include <stdio.h>
// пишем функцию для чтения текстового файла в список
void LoadPlayList ( HWND hListBox, const char* file)
{
    FILE* fList; // дескриптор для файла
    int index, ch;
    // открываем файл для чтения
    if ( ( fList = fopen ( file, "r")) != NULL)
    {
      // читаем данные, пока не достигнем конца файла
      while ( !feof ( fList))
      {
        // определяем буфер для данных
        char track[MAX_PATH] = " ";
        index = 0; // обнуляем счетчик
        // получаем первый символ
        ch = fgetc ( fList);
        // если это не конец файла и не перевод на следующую строку,
        while ( ( ch != '\n') && ( ch != EOF))
        {
          // копируем в буфер прочитанные символы
          track[index] = ( char) ch;
```

```
        // получаем следующий символ
        ch = fgetc ( fList);
        // увеличиваем счетчик
        index ++;
      }
      // добавляем строку в список
      SendMessage ( hListBox, LB_ADDSTRING, 0,
                    ( LPARAM) ( LPCTSTR) track);
      // переходим к следующей строке в файле
      track[index] = 0;
      ch = 0;
    }
    // закрываем файл
    fclose ( fList);
  }
}
```

Думаю, пример достаточно простой и не требует дополнительных комментариев. Замечу лишь, что функция feof необходима для определения конца файла.

Функция fputc позволяет выполнять посимвольную запись в файл. Кроме дескриптора открытого файла, в качестве аргумента функция принимает значение одиночного символа. В листинге 20.17 представлен пример для записи данных из стандартного списка в файл.

Листинг 20.17. Использование функции fputc

```
#include <stdio.h>
// пишем функцию для считывания данных из списка в текстовый файл
void SavePlayList ( HWND hListBox, const char* file)
{
   FILE* fSave; // дескриптор для файла
   char buf[MAX_PATH] = " "; // буфер для данных
   char* ptp; // указатель на строку
   // открываем файл для записи
   if ( ( fSave = fopen ( file,"w")) != NULL)
   {
     // определяем число элементов в списке
     int count = SendMessage ( hListBox, LB_GETCOUNT, 0,  0);
     // запускаем цикл обработки
     for ( int i = 0; i < count; i++)
     {
```

```
      // получаем строку из списка
      SendMessage ( hListBox, LB_GETTEXT, i,
                   ( LPARAM) ( LPCTSTR) buf);
      // устанавливаем указатель на строку
      ptr = buf;
      // записываем строку в файл
      while ( ( *ptr != '\0') && fputc ( *( ptr++), fSave) != EOF);
      // записываем в файл перевод строки
      fputc ( '\n', fSave);
    }
    // закрываем файл
    fclose ( fSave);
  }
}
```

Как видно из примера, перед записью в файл выполняются две проверки: на завершающий нулевой символ и ошибку записи (EOF). Это необходимо для исключения различных ошибок доступа. В остальном этот пример вполне доступен для понимания и не требует дополнительных пояснений.

Кроме чтения отдельных символов имеется возможность считывать или записывать за один раз целую строку с помощью функций fgets и fputs. Это позволяет немного упростить и улучшить код. Функция чтения имеет три аргумента: указатель на буфер данных, максимальное число считываемых символов и указатель на дескриптор открытого файла (FILE*). Функция записи определяет всего два аргумента: указатель на строку и дескриптор файла. В листинге 20.18 представлены примеры работы с данными функциями.

Листинг 20.18. Использование функций fgets и fputs

```
#include <stdio.h>
// демонстрационный пример для чтения строки из файла
void GetSrtingFile ( const char* file)
{
   FILE* fRead; // дескриптор для файла
   char buf[MAX_PATH] = " "; // буфер для данных
   // открываем файл для чтения
   if ( ( fRead = fopen ( file, "r")) != NULL)
   {
     // читаем строку
     while ( !feof ( fRead)) // пока не достигнут конец файла
     {
```

```
      // если строка прочитана, отображаем ее на экране
      if ( fgets ( buf, MAX_PATH, fRead) != NULL)
           MessageBox ( NULL, buf, file, MB_OK);
    }
    // закрываем файл
    fclose ( fRead);
  }
}
// пример функции для записи строк в файл из списка
void SavePlayList ( HWND hListBox, const char* file)
{
   FILE* fSave; // дескриптор для файла
   char buf[MAX_PATH] = " "; // буфер для данных
   // открываем файл для записи
   if ( ( fSave = fopen ( file,"w")) != NULL)
   {
     // определяем число элементов в списке
     int count = SendMessage ( hListBox, LB_GETCOUNT, 0,  0);
     // запускаем цикл обработки
     for ( int i = 0; i < count; i++)
     {
      // получаем строку из списка
      SendMessage ( hListBox, LB_GETTEXT, i,
                   ( LPARAM) ( LPCTSTR) buf);
      // записываем строку в файл
      fputs ( buf, fSave);
      // записываем в файл перевод строки
      fputc ( '\n', fSave};
     }
     // закрываем файл
     fclose ( fSave);
   }
```

И в завершение рассмотрим две полезные функции для записи или чтения из файла форматированных данных: `fprintf` и `fscanf`. Как правило, наиболее удобно применять их в паре, т. е. вначале выполнять запись данных определенного формата (`fprintf`) в файл, а затем чтение (`fscanf`) из этого файла по строго оговоренной схеме. Обе функции имеют два обязательных аргумента: указатель на дескриптор открытого файла и указатель на строку с форматируемыми данными. Кроме этого, в зависимости от решаемой задачи пользователь может добавлять собственные аргументы. Все данные преобразуются посредством специальных управляющих символов, перечисленных в табл. 20.1.

Общий формат представления одиночного значения следующий:

`% [ флаг ] [ ширина ] [ . точность ] [ { h | I64 | L | l } ] управляющий символ`

Необязательные параметры взяты в квадратные скобки. Значение флага позволяет корректировать отдельные параметры (табл. 20.2). Ширина определяет минимальное количество записываемых знаков и является положительным десятичным числом. Если количество знаков в форматируемом параметре меньше заданного, выходное значение будет дополнено пробелами или нулями. Для задания числа знаков после запятой применяется такой параметр, как точность. Он будет иметь смысл только для соответствующих типов данных (например, `float`). Управляющий символ представляет собой определенную букву латинского алфавита, определяющую конкретный тип данных.

Таблица 20.1. Управляющие символы

Код символа	Описание
`c` или `C`	Символ и широкий символ
`d` или `i`	Десятичное целое со знаком
`o`	Восьмеричное целое без знака
`u`	Десятичное целое без знака
`x` или `X`	Шестнадцатеричное целое без знака (строчные или прописные символы)
`e` или `E`	Экспоненциальное значение (строчные или прописные символы)
`f`	Значение с плавающей точкой
`n`	Указатель на целое значение, определяющее число символов, выведенных в поток на текущий момент
`p`	Адрес, на который указывает аргумент
`s` или `S`	Символьная строка

Таблица 20.2. Список флагов

Флаг	Описание
–	Выравнивание по левому краю
+	Добавление знака, если аргумент знаковый
#	Для шестнадцатеричных чисел выводится префикс 0x, а для восьмеричных — 0
0	Дополнение нулями до заданной ширины

В листинге 20.19 приведены различные примеры работы с данными функциями.

Листинг 20.19. Использование функций `fprintf` и `fscanf`

```
#include <stdio.h>
// переменные
FILE* fIni = NULL;
char* General =  "General Settings";
int x = 50, y = 78;
DWORD dwHex = 248945L;
char ch = 'Y';
DWORD dwValue = 9802348;
float fFreq = 2350.345;
// создаем новый текстовый файл
if ( ( fIni = fopen ( "F:\\MySettings.ini", "w")) != NULL)
{
   // записываем строку в файл
   fprintf ( fIni, "%s\n\n", General);
   // записываем целые значения в файл
   fprintf ( fIni, "Width = %i\nHeight = %d\n", x, y);
   // записываем шестнадцатеричное значение в файл
   fprintf ( fIni, "Memory Address = 0x%x\n", dwHex);
   // записываем символ в файл
   fprintf ( fIni, "Access = %c\n\n", ch);
   // записываем строку в файл
   fprintf ( fIni, "[Network Settings]\n\n");
   // записываем длинное целое значение в файл
   fprintf ( fIni, "Count = %lu\n", dwValue);
   // записываем число с плавающей точкой в файл
   fprintf ( fIni, "Frequence = %4.3f", fFreq);
}
// закрываем файл
fclose (fIni);
// открываем файл для чтения
if ( ( fIni = fopen ( "F:\\MySettings.ini", "r")) != NULL)
{
   // получаем значения из файла
   char buf[18] = " ";
   // читаем строку из файла
   fscanf ( fIni, "%s", buf);
   // читаем целые значения из файла
   fscanf ( fIni, "%i%d", &x, &y);
```

```
    // читаем шестнадцатеричное значение из файла
    fscanf ( fIni, "0x%x", &dwHex);
    // читаем символ из файла
    fscanf ( fIni, %c", &ch);
    // читаем длинное целое значение из файла
    fscanf ( fIni, "%lu", &dwValue);
    // читаем число с плавающей точкой из файла
    fscanf ( fIni, "%4.3f", &fFreq);
}
// закрываем файл
fclose (fIni);
```

В этом примере мы создали файл настроек (ini) и записали в него различные параметры. После открыли этот же файл для чтения и, получив необходимые значения, сохранили их в соответствующих переменных. Естественно, в реальной программе операции записи или чтения файла следует выполнять по мере необходимости.

20.2. Сжатые файлы

Несмотря на постоянное удешевление устройств хранения данных и увеличение их объема, программы сжатия файлов остаются достаточно популярными и широко используемыми во всем мире. И основной причиной этого является, конечно, обмен информацией в глобальной сети Интернет. Поэтому я и решил немного познакомить читателя с поддержкой сжатых файлов в Windows.

Интерфейс Win32 API содержит набор функций для декомпрессии файлов, сжатых с помощью программы Compress.exe: `LZOpenFile`, `LZCopy`, `LZClose`, `LZSeek`, `LZRead` и `GetExpandedName`.

Функция `LZOpenFile` позволяет создать, открывать и удалять указанный файл. Она имеет три аргумента: имя файла, указатель на структуру `OFSTRUCT` и тип выполняемой операции. Как правило, эта функция вызывается дважды: для открытия сжатого файла и для создания нового восстановленного файла. После того как оба файла открыты, применяется функция `LZCopy`, которая восстанавливает сжатые данные исходного файла и копирует их в конечный файл. По завершении работы следует закрыть используемые файлы с помощью функции `LZClose`. Если требуется восстановить только часть (части) сжатого файла, можно воспользоваться функциями `LZSeek` и `LZRead`. Также существует возможность получить оригинальное имя сжатого файла с помощью функции `GetExpandedName`. В листинге 20.20 представлен пример работы с функциями декомпрессии файлов. В опциях компоновщика не забудьте добавить ссылку на библиотеку Lz32.lib.

Листинг 20.20. Декомпрессия сжатых файлов

```
#include <lzexpand.h>
// пишем функцию для восстановления сжатого файла
bool UnCompressFile ( const char* src_file, const char* dest_file)
{
   HFILE hCompress = NULL, hRestore = NULL; // дескрипторы файлов
   OFSTRUCT ofSrc, ofDest; // структуры для описания файлов
   // открываем исходный файл для чтения
   hCompress = LZOpenFile ( src_file, &ofSrc, OF_READ);
   // создаем новый файл
   hRestore = LZOpenFile ( dest_file, &ofDest, OF_CREATE);
   // выполняем декомпрессию исходного файла
   if ( LZCopy ( hCompress, hRestore) < 0)
   {
     // произошла ошибка
     LZClose ( hCompress);
     LZClose( hRestore);
     return false; // выходим из функции с ошибкой
   }
   // закрываем оба файла
   LZClose ( hCompress);
   LZClose( hRestore);
   // выходим из функции
   return true;
}
// пишем функцию для получения оригинального имени сжатого файла
bool GetFileCompressName ( char* src_file, char* original_name)
{
   if ( GetExpandedName ( src_file, original_name) != 1)
        return false; // произошла ошибка
   // выходим из функции
   return true; // ошибок нет
}
```

А теперь попробуем написать собственный кодек для сжатия файлов любого типа. Основной принцип его работы будет построен на исключении повторяющихся символов в файле. Для упрощения работы создадим новый класс `Compress`. Необходимые файлы класса представлены в листингах 20.21 и 20.22.

Листинг 20.21. Файл Compress.h

```
#include <stdio.h>
// размер буфера
#define BUFFER_SIZE   32768U
```

```
// размеры данных
#define TMP_SIZE_4    4
#define TMP_SIZE_8    8
// объявления класса
class Compress
{
public:
   Compress (); // конструктор
   ~Compress (); // деструктор
// общие функции
   // функция для компрессии файла
   void ZipFile ( const char* in_File, const char* zip_File);
   // функция для декомпрессии файла
   void UnZipFile ( const char* zip_File, const char* out_File);
private:
   FILE* in_file, *out_file; // дескрипторы файлов
   char* buffer; // указатель на буфер
   // закрытые функции
   void _compress (); // сжать файл
   void _decompress (); // восстановить сжатый файл
   // вычислить 16-разрядное значение
   unsigned int _getValue ( unsigned char param1, unsigned char param2);
}; // окончание класса
```

Листинг 20.22. Файл Compress.cpp

```
#include "stdafx.h"
// реализация класса Compress
Compress :: Compress ()
{
   // инициализируем переменные
   in_file = NULL;
   out_file = NULL;
   // выделяем память под буфер
   buffer = new char [BUFFER_SIZE];
   if ( !buffer) return;
}
Compress :: ~Compress ()
{
   // освобождаем память
   if ( buffer) delete [] buffer;
   in_file = NULL;
   out_file = NULL;
}
```

```
unsigned int Compress :: _getValue ( unsigned char param1,
                                     unsigned char param2)
{
   return ( ( ( unsigned) param1 << 7) ^ param2);
}
void _compress ()
{
   if ( !in_file || !out_file) return;
   // определяем необходимые переменные
   unsigned char value=0;
   unsigned int index = 0;
   char tmp[TMP_SIZE_4];
   int ch, i, count = 0, pos = 0;
   char ch_1 = 0, ch_2 = 0;
   // заполняем буфер символом ASCII с кодом 32
   memset ( buffer, 32, BUFFER_SIZE);
   // читаем первый символ из входного файла
   ch = fgetc ( in_file);
   // обрабатываем файл, пока не достигнем конца
   while ( ch != EOF)
   {
     // вычисляем 16-разрядное значение
     index = _getValue ( ch_1, ch_2);
     // проверяем совпадение прочитанного символа с буфером
     if ( buffer[index] == ( char) ch)
     {
       // устанавливаем признак совпадения
       value = value ^ ( 1 << count);
     }
     else
     {
       // сохраняем символ в буфер
       buffer[index] = ( char) ch;
       // сохраняем символ во временном буфере
       tmp[pos++] = ( char) ch;
     }
     // проверяем заполнение временного буфера
     if ( ++count == TMP_SIZE_4)
     {
       // записываем признак в файл
       fputc ( ( char) value, out_file);
       // записываем в файл символы из временного буфера
       for ( i=0; i < pos; i++)
            fputc ( tmp[i], out_file);
```

```
      // обнуляем переменные перед следующим использованием
      count = 0;
      pos = 0;
      value = 0;
    }
    ch_1 = ch_2;
    ch_2 = ( char) ch;
    // читаем следующий символ из файла
    ch = fgetc ( in_file);
  }
  // если есть остатки, записываем их в файл
  if ( count)
  {
    // записываем управляющее значение
    fputc ( ( char) value, out_file);
    // записываем символы
    for ( i=0; i < pos; i++)
      fputc ( buf[i], out_file);
  }
}
void Compress :: _decompress ()
{
  if ( !in_file || !out_file) return;
  // определяем необходимые переменные
  unsigned char value=0;
  unsigned int index = 0;
  char ch_1 = 0, ch_2 = 0;
  int ch1, ch2, count = TMP_SIZE_4;
  // заполняем буфер символом ASCII с кодом 32
  memset ( buffer, 32, BUFFER_SIZE);
  // читаем первый символ из входного файла
  ch1 = fgetc ( in_file);
  // обрабатываем файл, пока не достигнем конца
  while ( ch1 != EOF)
  {
    // получаем признак совпадения
    value = ( unsigned char) ( char) ch1;
    // выполняем обработку для каждого бита признака
    for ( count = 0; count < TMP_SIZE_4; count++)
    {
      if ( value & ( 1 << count))
      {
        // вычисляем 16-разрядное значение
        index = _getValue ( ch_1, ch_2);
```

```
        // читаем символ из буфера
        ch2 = buffer[index];
      }
      else
      {
        // читаем следующий символ
        ch2 = fgetc ( in_file);
        // если конец файла, выходим из функции
        if ( ch2 == EOF) return;
        // вычисляем 16-разрядное значение
        index = _getValue ( ch_1, ch_2);
        // сохраняем символ в буфер
        buffer[index] = ( char) ch2;
      }
      // записываем восстановленный символ в конечный файл
      fputc ( ch2, out_file);
      // копируем значения
      ch_1 = ch_2;
      ch_2 = ch2;
    }
    // читаем следующий символ из файла
    ch1 = fgetc ( in_file);
  }
}
void Compress :: ZipFile ( const char* in_File, const char* zip_File)
{
  // открываем файл
  if ( ( in_file = fopen ( in_File, "rb")) == NULL)
        return; // не удалось открыть файл
  // создаем конечный файл
  if ( ( out_file = fopen ( zip_File, "wb")) == NULL)
  {
    fclose ( in_file);
    return; // не удалось открыть файл
  }
  // выполняем сжатие файла
  _compress ( in_file, out_file);
  // закрываем файлы
  fclose ( in_file);
  fclose ( out_file);
  in_file = NULL;
  out_file = NULL;
}
```

```
void UnZipFile ( const char* zip_File, const char* out_File)
{
   // открываем сжатый файл
   if ( ( in_file = fopen ( zip_File, "rb")) == NULL)
          return; // не удалось открыть файл
   // создаем конечный файл
   if ( ( out_file = fopen ( out_File, "wb")) == NULL)
   {
     fclose ( in_file);
     return; // не удалось открыть файл
   }
   // выполняем декомпрессию сжатого файла
   _decompress ( in_file, out_file);
   // закрываем файлы
   fclose ( in_file);
   fclose ( out_file);
   in_file = NULL;
   out_file = NULL;
}
```

У нас получился несколько примитивный, но вполне работоспособный класс, позволяющий сжимать любые типы файлов. Используемый алгоритм сжатия не является уникальным и достаточно широко известен, но для начинающих программистов он будет очень полезен как демонстрация одного из методов сжатия данных. К сожалению, его эффективность заметна только при работе с текстовыми файлами (txt, doc и др.).

20.3. Шифрование файлов

Для защиты файлов и каталогов от нежелательного доступа интерфейс Win32 API предусматривает возможность шифрования файловых объектов.

Если в операционной системе Windows 2000 установлена файловая система NTFS, можно организовать защиту с помощью следующих функций: `FileEncryptionStatus` (получение статуса файла), `EncryptFile` (шифровка файла), `DecryptFile` (дешифровка файла) и `EncryptionDisable` (управление защитой). Эти функции поддерживают шифрование не только файлов, но и каталогов. Если необходимо создать новый зашифрованный файл, достаточно вызвать функцию `CreateFile` с флагом `FILE_ATTRIBUTE_ENCRYPTED`.

Функция `FileEncryptionStatus` позволяет определить, является ли зашифрованным указанный файл. Первый аргумент определяет имя файла, а второй получает результат выполнения функции. Для проверки статуса файла из полученного значения выделяются флаги `FILE_IS_ENCRYPTED` (файл зашифрован) или `FILE_ENCRYPTABLE` (файл может быть зашифрован).

Функция EncryptFile выполняет шифровку заданного файла или каталога. Все вновь создаваемые файлы в защищенном каталоге будут зашифрованы. Единственный аргумент функции определяет имя файлового объекта. Для корректной работы функции требуется монопольный доступ к файловому объекту.

Функция DecryptFile восстанавливает зашифрованный файл или каталог. Она имеет два аргумента, первый из которых определяет имя файла или каталога, а второй зарезервирован. Для корректной работы функции требуется монопольный доступ к файловому объекту.

И последняя функция EncryptionDisable служит для блокировки каталога с файлами от нежелательной шифровки. Первый из двух аргументов функции указывает на путь к каталогу, а второй отключает или включает режим блокировки.

В листинге 20.23 представлены примеры работы с вышеупомянутыми функциями.

Листинг 20.23. Шифрование файлов в Windows 2000

```
#include <Windows.h>
#include <Winbase.h>
// пишем функцию для шифрования существующего файла
bool _encryptFile ( const char* file)
{
   DWORD dwStatus = 0;
   // проверяем статус файла
   if ( !FileEncryptionStatus ( file, &dwStatus))
        return false; // ошибка
   // выделяем значение FILE_IS_ENCRYPTED
   if ( dwStatus & FILE_IS_ENCRYPTED) == FILE_IS_ENCRYPTED)
        return true; // файл уже зашифрован
   // выполняем шифровку файла
   if ( !EncryptFile ( file))
        return false; // произошла ошибка
   // выходим из функции
   return true;
}
// пишем функцию для дешифровки существующего файла
bool _decryptFile ( const char* file)
{
   DWORD dwStatus = 0;
   // проверяем статус файла
   if ( !FileEncryptionStatus ( file, &dwStatus))
        return false; // ошибка
```

```
   // выделяем значение FILE_IS_ENCRYPTED
   if ( dwStatus & FILE_IS_ENCRYPTED) != FILE_IS_ENCRYPTED)
        return true; // файл не зашифрован
   // выполняем дешифровку файла
   if ( !DecryptFile ( file))
        return false; // произошла ошибка
   // выходим из функции
   return true;
}
// пишем функцию блокировки для указанного каталога
bool LockEncryptFiles ( LPCWSTR lpPath, bool bLock)
{
   if ( !EncryptionDisable ( lpPath, bLock))
        return false; // ошибка
   // выходим из функции
   return true;
}
```

Кроме представленных возможностей шифрования, существует отдельный интерфейс для защиты файловых объектов, который поддерживается всеми операционными системами Windows. Мы не будем рассматривать его целиком, поскольку для этого не хватит и всей книги, а разберем только базовые средства для шифрования различных типов файлов.

Создайте каркас нового класса и назовите его, например, `CryptoFile`. Заполните оба файла класса так, как это сделано в листингах 20.24 и 20.25. Не забудьте указать в опциях компоновщика ссылку на библиотеку Advapi32.lib.

Листинг 20.24. Файл CryptoFile.h

```
// подключаем определения специальных типов данных
typedef unsigned long HCRYPTPROV;
typedef unsigned long HCRYPTKEY;
typedef unsigned long HCRYPTHASH;
// типы кодеков, поддерживающие потоковую обработку файлов
#define CODEC_DES  CALG_DES
#define CODEC_RC2  CALG_RC2
#define CODEC_RC4  CALG_RC4
// размер блока данных
#define BLOCK_SIZE 8
// объявляем класс
class CryptoFile
{
```

```
public:
   CryptoFile (); // конструктор
   ~CryptoFile (); // деструктор
// общие функции
   // функция шифрования файлов любого типа
   bool Encrypt ( const char* src_file, const char* dest_file,
                  const char* Password, bool bCreateKey);
   // функция для дешифровки файлов любого типа
   bool Decrypt ( const char* src_file, const char* dest_file,
                  const char* Password, bool bCreateKey);
private:
   // переменные класса
   FILE* m_src_file; // указатель на файл источника
   FILE* m_dest_file; // указатель на файл назначения
   HCRYPTPROV m_Provider; // дескриптор доступа
   HCRYPTKEY m_Key; // дескриптор ключа
   HCRYPTHASH m_Hash; // дескриптор хэширования
// закрытые функции класса
   bool _createKey (); // функция для автоматической генерации ключа
   // функция для восстановления сгенерированного ключа
   bool _readKey ();
}; // окончание класса
```

Листинг 20.25. Файл CryptoFile.cpp

```
// необходимые файлы подключений
#include "stdafx.h"
#include <stdio.h>
#include <wincrypt.h>
#include "CryptoFile.h"
// реализация класса
CryptoFile :: CryptoFile ()
{
   // обнуляем переменные класса
   m_src_file = NULL;
   m_dest_file = NULL;
   m_Provider = 0ul;
   m_Key = 0ul;
   m_Hash = 0ul;
}
CryptoFile :: ~CryptoFile ()
{
   // освобождаем указатели
   m_src_file = NULL;
```

```
  m_dest_file = NULL;
}
bool CryptoFile :: _createKey ()
{
   HCRYPTKEY UserKey = 0ul; // дескриптор пользовательского ключа
   DWORD dwLenKey = 0; // длина ключа
   PBYTE pKey = NULL; // указатель на буфер данных
   // генерируем случайный ключ
   if ( !CryptGenKey ( m_Provider, CODEC_DES, CRYPT_EXPORTABLE,
                       &m_Key))
        return false; // ошибка
   // получаем дескриптор пользовательского ключа
   if ( !CryptGetUserKey ( m_Provider, AT_KEYEXCHANGE, &UserKey))
        return false; // ошибка
   // определяем размер ключа
   if ( !CryptExportKey (m_Key, UserKey, SIMPLEBLOB, 0, NULL,
                          &dwLenKey))
        return false; // ошибка
   // выделяем память
   pKey = ( BYTE*) malloc ( dwLenKey);
   // проверяем результат
   if ( !pKey)
   {
     // удаляем дескрипторы
     if ( UserKey) CryptDestroyKey  ( UserKey);
     return false; // ошибка
   }
   // шифруем ключ и экспортируем для рабочего ключа
   if ( !CryptExportKey ( m_Key, UserKey, SIMPLEBLOB, 0, pKey,
                          &dwLenKey))
   {
     // удаляем дескрипторы
     if ( UserKey) CryptDestroyKey  ( UserKey);
     return false; // ошибка
   }
   // удаляем пользовательский ключ
   CryptDestroyKey ( UserKey);
   UserKey = 0ul;
   // записываем размер ключа в файл назначения
   fwrite ( &dwLenKey, sizeof ( DWORD), 1, m_dest_file);
   // записываем значение ключа в файл
   fwrite ( pKey, sizeof ( DWORD), 1, m_dest_file);
   // освобождаем память
   if ( pKey) free ( pKey);
```

```
   // выходим из функции
   return true;
}
bool CryptoFile :: _readKey ()
{
   DWORD dwLenKey = 0; // длина ключа
   PBYTE pKey = NULL; // указатель на буфер данных
   // читаем из исходного файла значение длины ключа
   fread ( &dwLenKey, sizeof ( DWORD), 1, m_src_file);
   // если обнаружен конец файла, выходим
   if ( feof ( m_src_file)) return false;
   // выделяем память для значения ключа
   pKey = ( BYTE*) malloc ( dwLenKey);
   // если отказано в выделении памяти, выходим
   if ( !pKey) return false;
   // читаем значение ключа из исходного файла
   fread ( pKey, 1,  dwLenKey, m_src_file);
   // если обнаружен конец файла, выходим
   if ( feof ( m_src_file)) return false;
   // восстанавливаем дескриптор ключа
   if ( !CryptImportKey ( m_Provider, pKey, dwLenKey, 0, 0, &m_Key))
   {
     // освобождаем память
     if ( pKey) free ( pKey);
          return false; // ошибка
   }
   // освобождаем память
   if ( pKey) free ( pKey);
   // выходим из функции
   return true;
}
bool CryptoFile :: Encrypt ( const char* src_file, const char* dest_file,
                              const char* Password, bool bCreateKey)
{
   DWORD dwDataLen = 0, dwBufferLen = 0, dwReadBytes = 0;
   PBYTE pBuffer = NULL; // указатель на буфер данных
   bool bResult = true; // результат операции
   // открываем исходный файл
   if ( (  m_src_file  = fopen ( src_file, "rb")) == NULL)
      return false; // ошибка
   // открываем конечный файл
   if ( ( m_dest_file  = fopen ( dest_file, "wb")) == NULL)
   {
      fclose ( m_src_file);
```

```
    return false; // ошибка
  }
  // получаем дескриптор доступа по умолчанию
  if ( !CryptAcquireContext ( &m_Provider, NULL, NULL, PROV_RSA_FULL,
                             0))
  {
    bResult = false; // ошибка
    goto exit_error;
  }
  // если требуется, генерируем ключ
  if ( bCreateKey)
  {
    if ( !_createKey ())
    {
      bResult = false; // ошибка
      goto exit_error;
    }
  }
  else
  {
    // инициализируем дескриптор хэширования для потока данных
    if ( !CryptCreateHash ( m_Provider, CALG_MD5, 0, 0, &m_Hash))
    {
      bResult = false; // ошибка
      goto exit_error;
    }
    // обрабатываем заданный пароль
    if ( !CryptHashData ( m_Hash, ( BYTE*) Password,
                         strlen ( Password), 0))
    {
      bResult = false; // ошибка
      goto exit_error;
    }
    // генерируем ключ на основе пароля
    if ( !CryptDeriveKey ( m_Provider, CODEC_DES, m_Hash, 0, &m_Key))
    {
      bResult = false; // ошибка
      goto exit_error;
    }
    // удаляем дескриптор хэширования
    CryptDestroyHash ( m_Hash);
    m_Hash = 0ul;
  }
```

```
    // вычисляем число байтов в одном блоке данных
    dwDataLen = 1000 - 1000 % BLOCK_SIZE;
    // вычисляем размер блока данных
    if ( BLOCK_SIZE > 1)
      dwBufferLen = dwDataLen + BLOCK_SIZE;
    else
      dwBufferLen = dwDataLen;
    // выделяем память
    pBuffer  = ( BYTE*) malloc ( dwBufferLen);
    // проверяем результат
    if ( !pBuffer)
    {
      bResult = false; // ошибка
      goto exit_error;
    }
    // шифруем исходный файл
    do
    {
      // читаем блок данных из исходного файла
      dwReadBytes = fread ( pBuffer, 1, dwDataLen, m_src_file);
      // шифруем блок данных
      if ( !CryptEncrypt ( m_Key, 0, feof ( m_src_file), 0, pBuffer,
                           &dwReadBytes, dwBufferLen))
      {
        bResult = false; // ошибка
        goto exit_error;
      }
      // пишем зашифрованный блок данных в конечный файл
      fwrite ( pBuffer, 1, dwReadBytes, m_dest_file);
    // переходим к чтению следующего блока данных
    } while ( !feof ( m_src_file));
// метка для обработки выхода из функции
exit_error:
    // освобождаем память
    if ( pBuffer) free ( pBuffer);
    // закрываем файлы
    if ( m_src_file) fclose ( m_src_file);
    if ( m_dest_file) fclose ( m_dest_file);
    // удаляем ключ
    if ( m_Key) CryptDestroyKey  (m_Key);
    // удаляем дескриптор хэширования
    if ( m_Hash) CryptDestroyHash ( m_Hash);
```

```
   // удаляем дескриптор доступа
   if ( m_Provider) CryptReleaseContext ( m_Provider, 0);
   // обнуляем переменные
   m_src_file = NULL;
   m_dest_file = NULL;
   m_Provider = 0ul;
   m_Key = 0ul;
   m_Hash = 0ul;
   // выходим из функции
   return  bResult;
}
bool CryptoFile :: Decrypt (const char* src_file, const char* dest_file,
                            const char* Password, bool bCreateKey)
{
   DWORD dwDataLen = 0, dwBufferLen = 0, dwReadBytes = 0;
   PBYTE pBuffer = NULL; // указатель на буфер данных
   bool bResult = true; // результат операции
   // открываем исходный файл
   if ( (  m_src_file  = fopen ( src_file, "rb")) == NULL)
      return false;
   // открываем конечный файл
   if ( ( m_dest_file  = fopen ( dest_file, "wb")) == NULL)
   {
     fclose ( m_src_file);
     return false; // ошибка
   }
   // получаем дескриптор доступа по умолчанию
   if ( !CryptAcquireContext ( &m_Provider, NULL, NULL, PROV_RSA_FULL,
                               0))
   {
     bResult = false; // ошибка
     goto exit_error;
   }
   // если требуется, генерируем ключ
   if ( bCreateKey)
   {
     if ( !_readKey ())
     {
       bResult = false; // ошибка
       goto exit_error;
     }
   }
```

```
else
{
  // инициализируем дескриптор хэширования для потока данных
  if ( !CryptCreateHash ( m_Provider, CALG_MD5, 0, 0, &m_Hash))
  (
     bResult = false; // ошибка
     goto exit_error;
  }
  // обрабатываем заданный пароль
  if ( !CryptHashData ( m_Hash, ( BYTE*) Password,
                        strlen ( Password), 0))
  {
    bResult = false; // ошибка
    goto exit_error;
  }
  // генерируем ключ на основе пароля
  if ( !CryptDeriveKey ( m_Provider, CODEC_DES , m_Hash, 0,
                        &m_Key))
  {
    bResult = false; // ошибка
    goto exit_error;
  }
  // удаляем дескриптор хэширования
  CryptDestroyHash ( m_Hash);
  m_Hash = 0ul;
}
// вычисляем число байтов для шифровки за один проход
dwDataLen = 1000 - 1000 % BLOCK_SIZE;
// размер блока данных
dwBufferLen = dwDataLen;
// выделяем память
pBuffer  = ( BYTE*) malloc ( dwBufferLen);
// проверяем результат
if ( !pBuffer)
{
  bResult = false; // ошибка
  goto exit_error;
}
// дешифруем исходный файл
do
{
  // читаем блок данных из исходного файла
  dwReadBytes = fread ( pBuffer, 1, dwDataLen, m_src_file);
```

```
    // дешифруем блок данных
    if ( !CryptDecrypt ( m_Key, 0, feof ( m_src_file), 0, pBuffer,
                         &dwReadBytes))
    {
      bResult = false; // ошибка
      goto exit_error;
    }
    // пишем дешифрованный блок данных в конечный файл
    fwrite ( pBuffer, 1, dwReadBytes, m_dest_file);
    // переходим к чтению следующего блока данных
  } while ( !feof ( m_src_file));
// метка для обработки выхода из функции
exit_error:
  // освобождаем память
  if ( pBuffer) free ( pBuffer);
  // закрываем файлы
  if ( m_src_file) fclose ( m_src_file);
  if ( m_dest_file) fclose ( m_dest_file);
  // удаляем ключ
  if ( m_Key) CryptDestroyKey  (m_Key);
  // удаляем дескриптор хэширования
  if ( m_Hash) CryptDestroyHash ( m_Hash);
  // удаляем дескриптор доступа
  if ( m_Provider) CryptReleaseContext ( m_Provider, 0);
  // обнуляем переменные
  m_src_file = NULL;
  m_dest_file = NULL;
  m_Provider = 0ul;
  m_Key = 0ul;
  m_Hash = 0ul;
  // выходим из функции
  return  bResult;
}
```

Прежде чем компилировать созданный класс, откройте файл Stdafx.h и добавьте после `#define WIN32_LEAN_AND_MEAN` указатель версии в виде строки `#define _WIN32_WINNT 0x0400`. Если вы используете оболочку Visual Studio .NET, делать это не обязательно.

Класс `CryptoFile` поддерживает три алгоритма шифрования: DES (56-битный ключ), RC2 (128-битный ключ) и RC4 (128-битный ключ). Для обработки файлов используются две функции: `Encrypt` и `Decrypt`. Они имеют по четыре схожих аргумента: имена исходного и конечного файлов, указатель

на установленный пользователем пароль и режим генерации ключа. Если с первыми тремя аргументами все ясно, то последний требует пояснения. С его помощью можно определять способ создания ключа. При установке значения этого параметра в TRUE заданный пароль не будет использоваться, а ключ будет сгенерирован автоматически, иначе ключ составится на основе введенного пользователем пароля. Создание и получение автоматически сгенерированного ключа выполняют функции _createKey и _readKey.

Шифрование файлов организовано с помощью следующих библиотечных функций: CryptAcquireContext (получение дескриптора для доступа к определенным криптоалгоритмам), CryptCreateHash (инициализация и получение дескриптора хэширования для потоковых данных), CryptHashData (добавление данных пользователя к объекту хэширования), CryptDeriveKey (генерация криптографического ключа на основе данных объекта хэширования), CryptDestroyHash (освобождение дескриптора хэширования), CryptEncrypt (шифрование данных по указанному алгоритму и ключу), CryptDestroyKey (освобождение дескриптора ключа), CryptReleaseContext (освобождение дескриптора доступа), CryptDecrypt (дешифровка данных по указанному алгоритму и ключу), CryptGenKey (генерация случайного криптографического ключа), CryptGetUserKey (получение дескриптора для пользовательского ключа), CryptExportKey (экспорт криптографического ключа или пары ключей для алгоритма шифрования) и CryptImportKey (передача криптографического ключа объекту доступа).

И в завершение темы изучите пример работы с классом CryptoFile в листинге 20.26.

Листинг 20.26. Использование класса CryptoFile в программе

```
// подключаем файл определений класса
#include "ryptoFile.h"
// создаем объект класса
CryptoFile crypt;
// определяем пароль для шифрования файла
char* pass = "mycryptclass";
// выполняем шифровку файла с заданным паролем
crypt.Encrypt ( "C:\\MySecret.doc", "C:\\MySecret.des", pass, false);
// выполняем дешифровку файла с заданным паролем
crypt.Decrypt ( "C:\\MySecret.des", "C:\\MySecret.doc", pass, false);
// выполняем шифровку файла в режиме автоматической генерации пароля
crypt.Encrypt ( "C:\\MySecret.doc", "C:\\MySecret.des", NULL, true);
// выполняем дешифровку файла в режиме автоматической генерации пароля
crypt.Decrypt ( "C:\\MySecret.des", "C:\\MySecret.doc", NULL, true);
```

По умолчанию для шифрования файлов используется алгоритм DES, но вы можете самостоятельно выбрать RC2 (CODEC_RC2) или RC4 (CODEC_RC4), выбрав соответствующую константу и заменив ее в коде класса. Я же, в свою очередь, хочу на этом завершить вопросы программирования файловой системы в Windows и надеюсь, что рассмотренный материал поможет вам не только в повседневной работе, но и в дальнейшем изучении поистине безграничных глубин интерфейса Win32 API.

ГЛАВА 21

Работа с Интернетом

Думаю, не стоит говорить, какое значение имеет глобальная сеть в современной жизни. Трудно представить современную динамически развивающуюся организацию без компьютерной техники и средств доступа в Интернет. Непременным условием стабильной работы является организация собственных ресурсов в сети для рекламирования услуг, а также для предоставления обновленной информации потенциальным и реальным пользователям о производимой продукции. Все это требует не только желания, но и соответствующего программного обеспечения. Именно по этой причине сегодняшнему разработчику так необходимо иметь представление о сетевых протоколах и интерфейсах, позволяющих создавать современные коммерческие программные продукты. Несмотря на разнообразие всевозможных скриптовых языков, все базовые компоненты пишутся исключительно на C/C++. В этой главе мы поговорим о некоторых важных вопросах программирования, связанных со Всемирной паутиной. Мне бы хотелось выделить несколько базовых тем, которые будут рассмотрены далее:

1. Создание соединения для доступа в сеть.
2. Просмотр интернет-ресурсов или организация собственного браузера.
3. Загрузка файлов.

21.1. Создание нового соединения

Как известно, прежде чем войти в сеть с помощью модема (или другого устройства), требуется создать специальное соединение. В Windows оно представляет собой файловый объект, имеющий определенный формат и свойства. Двойной щелчок на таком объекте загружает необходимые библиотеки и ресурсы системы, позволяя установить удаленное модемное соединение с сервером провайдера интернет-услуг и через него выйти в гло-

бальную сеть. Можно, конечно, воспользоваться специальным мастером создания соединения, но мы пойдем более "сложным" путем и выполним эту задачу программно.

Чтобы получить новое соединение для удаленного доступа, нужно применить программный интерфейс RAS (Remote Access Service — служба удаленного доступа). Все необходимое находится в системной библиотеке Rasapi32.dll, поэтому мы создадим класс, динамически загружающий данный модуль, а затем получим указатели на необходимые нам функции. Пример реализации такого класса показан в листингах 21.1 и 21.2.

Листинг 21.1. Файл Connect.h класса Connect

```
// подключаем необходимый файл определений
#include <ras.h>
// объявляем указатели на используемые библиотечные функции
// указатель на функцию RasEnumDevices
typedef DWORD ( CALLBACK* pfnRASENUMDEVICES) ( LPRASDEVINFO lpDevInfo,
                LPDWORD lpcb, LPDWORD lpcDevices);
// указатель на функцию RasValidateEntryName
typedef DWORD ( CALLBACK* pfnRASVALIDATEENTRYNAME) ( LPCTSTR lpszBook,
                LPCTSTR lpszEntry);
// указатель на функцию RasSetEntryProperties
typedef DWORD ( CALLBACK* pfnRASSETENTRYPROPERTIES) ( LPCTSTR lpszBook,
      LPCTSTR lpszEntry,   LPRASENTRY lpRasEntry, DWORD dwEntryInfoSize,
      LPBYTE lpbDeviceInfo,   DWORD dwDeviceInfoSize);
// указатель на функцию RasSetEntryDialParams
typedef DWORD ( CALLBACK* pfnRASSETENTRYDIALPARAMS) ( LPCTSTR lpszBook,
                LPRASDIALPARAMS lprasdialparams, BOOL fRemovePassword);
// объявляем наш класс
class Connect
{
public:
   Connect (); // конструктор
   ~Connect (); // деструктор
// общие функции
   // перечисление доступных устройств
   bool EnumDevices ( HWND hComboBox);
   // функция для создания нового соединения
   bool CreateConnect ( const char* device, const char* name,
     const char* phone_number, const char* user, const char* password);
private:
   HINSTANCE hRasLibrary; // дескриптор системной библиотеки Rasapi32.dll
```

```
// закрытые функции
   // перечисление доступных совместимых устройств
   DWORD _RasEnumDevices ( LPRASDEVINFO lpRasDevInfo, LPDWORD lpcb,
                           LPDWORD lpcDevices);
   // проверка правильности формата имени соединения
   DWORD _RasValidateEntryName ( LPCTSTR lpszPhonebook,
                                 LPCTSTR lpszEntry);
   // создание и модификация новой записи в телефонной книге
   DWORD _RasSetEntryProperties ( LPCTSTR lpszPhonebook,
         LPCTSTR lpszEntry, LPRASENTRY lpRasEntry, DWORD dwEntryInfoSize,
         LPBYTE lpbDeviceInfo, DWORD dwDeviceInfoSize);
   // редактирование информации о текущем соединении
   DWORD _RasSetEntryDialParams ( LPCTSTR lpszPhonebook,
              LPRASDIALPARAMS lprasdialparams, BOOL fRemovePassword);
}; // окончание класса
```

Листинг 21.2. Файл Connect.cpp класса Connect

```
// подключаем необходимые файлы
#include "stdafx.h"
#include "Connect.h"
// реализация класса Connect
Connect :: Connect ()
{
   // обнуляем переменные
   hRasLibrary = NULL;
   // загружаем системную библиотеку
   hRasLibrary = LoadLibrary ( TEXT ( "rasapi32.dll"));
   // если библиотека не загружена, выходим
   if ( !hRasLibrary) return;
}
Connect :: ~Connect ()
{
   // выгружаем системную библиотеку
   if ( hRasLibrary) FreeLibrary ( hRasLibrary);
}
DWORD Connect : :_RasEnumDevices ( LPRASDEVINFO lpRasDevInfo,
                                   LPDWORD lpcb, LPDWORD lpcDevices)
{
   pfnRASENUMDEVICES result;
   // получаем указатель на функцию
   result = ( pfnRASENUMDEVICES) GetProcAddress ( hRasLibrary,
                                                  "RasEnumDevices");
```

```
    // если не удалось получить указатель, выходим с ошибкой
    if ( !result) return 1;
    // иначе возвращаем указатель на функцию RasEnumDevices
    return ( *result) ( lpRasDevInfo,  lpcb, lpcDevices);
}
DWORD Connect : :_RasValidateEntryName ( LPCTSTR lpszPhonebook,
                                         LPCTSTR lpszEntry)
{
    pfnRASVALIDATEENTRYNAME result;
    // получаем указатель на функцию
    result = ( pfnRASVALIDATEENTRYNAME) GetProcAddress ( hRasLibrary,
                                         "RasValidateEntryName");
    // если не удалось получить указатель, выходим с ошибкой
    if ( !result) return 1;
    // иначе возвращаем указатель на функцию RasValidateEntryName
    return ( *result) ( lpszPhonebook, lpszEntry);
}
DWORD Connect :: _RasSetEntryProperties ( LPCTSTR lpszPhonebook,
       LPCTSTR lpszEntry, LPRASENTRY lpRasEntry, DWORD dwEntryInfoSize,
       LPBYTE lpbDeviceInfo, DWORD dwDevInfoSize)
{
    pfnRASSETENTRYPROPERTIES result;
    // получаем указатель на функцию
    result = ( pfnRASSETENTRYPROPERTIES) GetProcAddress ( hRasLibrary,
                                         "RasSetEntryProperties");
    // если не удалось получить указатель, выходим с ошибкой
    if ( !result) return 1;
    // иначе возвращаем указатель на функцию RasSetEntryProperties
    return ( *result) ( lpszPhonebook, lpszEntry, lpRasEntry,
                        dwEntryInfoSize, lpbDeviceInfo,  dwDevInfoSize);
}
DWORD Connect :: _RasSetEntryDialParams ( LPCTSTR lpszPhonebook,
                  LPRASDIALPARAMS lprasdialparams, BOOL fRemovePassword)
{
    pfnRASSETENTRYDIALPARAMS result;
    // получаем указатель на функцию
    result = ( pfnRASSETENTRYDIALPARAMS) GetProcAddress ( hRasLibrary,
                                         "RasSetEntryDialParamsA");
    // если не удалось получить указатель, выходим с ошибкой
    if (!result) return 0;
    // иначе возвращаем указатель на функцию RasSetEntryDialParams
    return ( *result) ( lpszPhonebook, lprasdialparams,
                     fRemovePassword);
}
```

```
bool Connect :: EnumDevices ( HWND hComboBox)
{
   DWORD dwBufferSize = 0, dwNum = 0; // размер буфера и число структур
   RASDEVINFO* pInfo = NULL; // указатель на информационную структуру
   // определяем необходимый размер буфера для структур RASDEVINFO
   _RasEnumDevices ( NULL, &dwBufferSize, &dwNum);
   // выделяем необходимую память из кучи и инициализируем нулями
   pInfo = ( LPRASDEVINFO) GlobalAlloc ( GPTR, dwBufferSize);
   // если память не выделена, выходим из функции
   if ( pInfo == NULL) return false;
   // определяем размер структуры
   pInfo->dwSize = sizeof ( RASDEVINFO);
   // перечисляем совместимые с RAS устройства
   if ( _RasEnumDevices ( pInfo, &dwBufferSize, &dwNum))
        return false; // выходим, если произошла ошибка
   // добавляем имена найденных устройств в комбинированный список
   for ( DWORD i = 0; i < dwNum; i++)
   {
     SendMessage ( hComboBox, CB_ADDSTRING, 0,
                         ( LPARAM) ( LPCTSTR) pInfo->szDeviceName);
     // переходим к следующему устройству
     pInfo++;
   }
   // освобождаем память
   GlobalFree ( pInfo);
   // выходим из функции
   return true;
}
bool Connect :: CreateConnect ( const char* device, const char* name,
     const char* phone_number, const char* user, const char* password)
{
   RASENTRY ras; // описание записи в телефонной книге
   // обнуляем структуру
   ZeroMemory ( &ras, sizeof ( RASENTRY));
   // проверяем правильность формата имени соединения
   if ( _RasValidateEntryName ( NULL, name) != ERROR_SUCCESS)
        return false; // выходим из функции, если ошибка
   // заполняем требуемые поля структуры RASENTRY
   ras.dwSize = sizeof ( RASENTRY); // размер структуры
   // стандартный путь для IP-пакетов
   ras.dwfOptions = RASEO_RemoteDefaultGateway;
   ras.dwfNetProtocols = RASNP_Ip; // сетевой протокол TCP/IP
   ras.dwFramingProtocol = RASFP_Ppp; // тип удаленного доступа PPP
```

```
    strcpy ( ras.szAreaCode, " "); // код города
    strcpy ( ras.szLocalPhoneNumber, phone_number); // номер телефона
    // полный путь к библиотеке (файл DLL) автодозвонки
    strcpy ( ras.szAutodialDll, " ");
    strcpy ( ras.szAutodialFunc, " "); // имя функции автодозвонки
    strcpy ( ras.szDeviceType, RASDT_Modem); // тип устройства
    strcpy ( ras.szDeviceName, device); // имя устройства
    // добавляем новую запись в телефонную книгу
    if ( _RasSetEntryProperties ( NULL, name, &ras, sizeof ( RASENTRY),
                                  NULL, NULL))
        return false; //выходим из функции, если ошибка
    // определяем настройки для удаленного соединения
    RASDIALPARAMS dial;
    // заполняем поля структуры RASDIALPARAMS
    dial.dwSize = sizeof ( RASDIALPARAMS); // размер структуры
    strcpy ( dial.szEntryName, name); // имя соединения
    strcpy (dial.szPhoneNumber, " "); // телефонный номер
    // телефонный номер обратного вызова
    strcpy ( dial.szCallbackNumber, " ");
    strcpy ( dial.szUserName, user); // имя пользователя
    strcpy ( dial.szPassword, password); // пароль пользователя
    // сохраняем настройки для удаленного соединения
    if ( _RasSetEntryDialParams ( NULL, &dial, false))
        return false; // выходим из функции, если ошибка
    // выходим из функции
    return true;
}
```

Созданный класс Connect содержит всего две открытые функции, одна из которых перечисляет доступные устройства, а другая создает новое соединение на основе введенных пользователем данных. Для реализации задачи мы задействовали всего четыре библиотечных функции: RasEnumDevices (перечисление устройств), RasValidateEntryName (проверка формата введенного пользователем имени соединения), RasSetEntryProperties (добавление записи в телефонную книгу) и RasSetEntryDialParams (настройка параметров удаленного соединения). Кроме функций были использованы специальные структуры: RASDEVINFO (информация об устройстве TAPI, совместимом с RAS), RASENTRY (описание формата записи в телефонной книге) и RASDIALPARAMS (описание параметров для настройки удаленного соединения).

Перед применением класса Connect в своей программе подготовьте в редакторе ресурсов пользовательский интерфейс: средства ввода данных и кнопку создания соединения. Добавьте раскрывающийся список, если необходимо предоставить пользователю возможность выбора устройства связи. После этого вызовите класс Connect, как показано в листинге 21.3.

Листинг 21.3. Использование класса Connect

```
#include "Connect.h"
// примерный код работы с классом Connect
Connect NewConnect; // объявляем класс
char device[150] = " "; // переменная для хранения имени устройства связи
// получаем дескриптор комбинированного списка
HWND hComboBox = GetDlgItem ( hDlg, IDC_MYCOMBO);
// перечисляем доступные устройства и добавляем их в список
NewConnect.EnumDevices ( hComboBox);
// устанавливаем первое найденное устройство по умолчанию
SendMessage ( hComboBox, CB_SETCURSEL, 0, 0);
// получаем номер и имя выбранного в списке устройства
int sel = SendMessage ( hComboBox, CB_GETCURSEL, ( WORD)0, 0L);
SendMessage ( hComboBox, CB_GETLBTEXT, ( WORD) sel, ( LONG) device);
// создаем новое соединение по умолчанию
NewConnect.CreateConnect ( device, "Test Connection", "123456", "MyName",
                           "MyPassword");
```

Как видите, создать новое соединение очень просто. Если вам понадобится более тонкая настройка параметров соединения, самостоятельно изучите документацию на представленные в классе `Connect` библиотечные функции и структуры.

21.2. Просмотр сетевых ресурсов

Всем хорошо известны наиболее популярные программы для просмотра интернет-страниц: Internet Explorer, Netscape Navigator, Opera. Основной принцип работы этих браузеров заключается в отображении сетевых ресурсов на экране пользователя и удобной системе навигации. Дополнительно каждое из перечисленных приложений предоставляет целый ряд сервисных услуг для настройки удобного интерфейса, защиты и языковых кодировок.

Мы попытаемся написать собственный простейший браузер под Windows, который будет выполнять только самые необходимые функции. Для решения поставленной задачи существует несколько базовых методов: библиотека MFC (класс `CHtmlView`), библиотека активных шаблонов ATL и Win32 API. Первый способ прост в реализации, но автоматически "цепляет" за собой множество файлов. Второй способ очень элегантен и красив, но не относится к теме книги, поэтому мы будем решать задачу с помощью самого сложного (на первый взгляд) варианта и применим всю мощь интерфейса Win32 API.

Уважаемый читатель знает, что наряду с огромным набором функций Win32 API содержит не один десяток интерфейсов, определяющих основные функ-

циональные возможности оболочки Windows. Для тех, кто не знаком с понятием интерфейса, поясню, что он представляет собой абстрактную оболочку, включающую в себя набор методов и свойств, организующих решение конкретной задачи. Имена всех интерфейсов начинаются с латинской буквы 'I', чтобы их можно было легко отличить от классов. И это важно помнить и понимать, поскольку интерфейс и класс — абсолютно разные понятия. Класс содержит в себе как определение, так и обязательную реализацию всех объявленных функций (здесь не идет речь об абстрактных классах и виртуальных функциях). Интерфейс всегда содержит определение методов (те же функции), но никогда их реализацию. Он служит в качестве обертки для реальных объектов. Все существующие и любые новые интерфейсы являются расширениями базового интерфейса `IUnknown`. Он имеет всего три метода (`QueryInterface`, `AddRef` и `Release`), которые обязательно добавляются к набору методов производного интерфейса. Нас в первую очередь будет интересовать метод `QueryInterface`, позволяющий получить указатель на требуемый интерфейс. Важно понимать, что интерфейс нельзя объявить как класс, поскольку он, в принципе, не существует (абстрактен), поэтому всегда используется только указатель на этот интерфейс. При наличии реально существующего в системе объекта метод `QueryInterface` возвратит указатель на него. Кроме того, программист должен написать реализацию всех методов как непосредственно используемого интерфейса, так и базовых (обычно это методы интерфейса `IUnknown`). Понятно, что многие определяемые методы не нужны, поэтому для них пишутся "заглушки". Как все это делается, вы увидите дальше. В любом случае, если вы ранее не сталкивались с понятием интерфейса, объектной модели COM и технологии OLE, имеет смысл почитать соответствующую литературу, например [5].

А теперь перейдем непосредственно к решению поставленной задачи. Нам понадобится написать реализацию для следующих интерфейсов: `IStorage` (поддержка объектов в памяти), `IOleClientSite` (отображение информации и интерфейса на экране), `IOleInPlaceSite` (управление связью между объектом и контейнером), `IOleInPlaceFrame` (управление окном верхнего уровня) и `IDocHostUIHandler` (обработка меню и панелей инструментов).

Прежде всего, создайте новый пустой проект типа **Win32 Application** и назовите его, например, **WebBrowser**. Для реализации каждого перечисленного интерфейса понадобятся отдельные заголовочные файлы: IStorage.h (листинг 21.4), IOleClientSite.h (листинг 21.5), IOleInPlaceSite.h (листинг 21.6), IOleInPlaceFrame.h (листинг 21.7) и IDocHostUIHandler.h (листинг 21.8).

Листинг 21.4. Реализация интерфейса `IStorage`

```
// список методов интерфейса IStorage
HRESULT STDMETHODCALLTYPE Stor_QueryInterface ( IStorage FAR* Stor,
 REFIID riid, LPVOID FAR* ppvObj);
```

```
HRESULT STDMETHODCALLTYPE Stor_AddRef ( IStorage FAR* Stor);
HRESULT STDMETHODCALLTYPE Stor_Release ( IStorage FAR* This);
HRESULT STDMETHODCALLTYPE Stor_CreateStream ( IStorage FAR* Stor,
 const WCHAR* pwcsName, DWORD grfMode, DWORD reserved1, DWORD reserved2,
 IStream** ppstm);
HRESULT STDMETHODCALLTYPE Stor_OpenStream ( IStorage FAR* Stor,
 const WCHAR* pwcsName, void* reserved1, DWORD grfMode, DWORD reserved2,
 IStream** ppstm);
HRESULT STDMETHODCALLTYPE Stor_CreateStorage(IStorage FAR* Stor,
 const WCHAR *pwcsName, DWORD grfMode, DWORD reserved1, DWORD reserved2,
 IStorage **ppstg);
HRESULT STDMETHODCALLTYPE Stor_OpenStorage ( IStorage FAR* Stor,
 const WCHAR* pwcsName, IStorage* pstgPriority, DWORD grfMode,
 SNB snbExclude, DWORD reserved, IStorage** ppstg);
HRESULT STDMETHODCALLTYPE Stor_CopyTo ( IStorage FAR* Stor,
 DWORD ciidExclude, IID const* rgiidExclude, SNB snbExclude,
 IStorage* pstgDest);
HRESULT STDMETHODCALLTYPE Stor_MoveElementTo ( IStorage FAR* Stor,
 const OLECHAR* pwcsName, IStorage* pstgDest, const OLECHAR* pwcsNewName,
 DWORD grfFlags);
HRESULT STDMETHODCALLTYPE Stor_Commit(IStorage FAR* Stor,
 DWORD grfCommitFlags);
HRESULT STDMETHODCALLTYPE Stor_Revert ( IStorage FAR* Stor);
HRESULT STDMETHODCALLTYPE Stor_EnumElements ( IStorage FAR* Stor,
 DWORD reserved1, void* reserved2, DWORD reserved3,
 IEnumSTATSTG** ppenum);
HRESULT STDMETHODCALLTYPE Stor_DestroyElement ( IStorage FAR* Stor,
 const OLECHAR* pwcsName);
HRESULT STDMETHODCALLTYPE Stor_RenameElement ( IStorage FAR* Stor,
 const WCHAR* pwcsOldName, const WCHAR* pwcsNewName);
HRESULT STDMETHODCALLTYPE Stor_SetElementTimes ( IStorage FAR* Stor,
 const WCHAR* pwcsName, FILETIME const* pctime, FILETIME const* patime,
 FILETIME const* pmtime);
HRESULT STDMETHODCALLTYPE Stor_SetClass ( IStorage FAR* Stor,
 REFCLSID clsid);
HRESULT STDMETHODCALLTYPE Stor_SetStateBits ( IStorage FAR* This,
 DWORD grfStateBits, DWORD grfMask);
HRESULT STDMETHODCALLTYPE Stor_Stat ( IStorage FAR* Stor,
 STATSTG* pstatstg, DWORD grfStatFlag);
// определяем порядок методов для виртуальной таблицы IStorage
IStorageVtbl IStorageMetods =
{
 Stor_QueryInterface, Stor_AddRef,Stor_Release, Stor_CreateStream,
 Stor_OpenStream, Stor_CreateStorage, Stor_OpenStorage, Stor_CopyTo,
```

```
 Stor_MoveElementTo, Stor_Commit, Stor_Revert, Stor_EnumElements,
 Stor_DestroyElement, Stor_RenameElement, Stor_SetElementTimes,
 Stor_SetClass, Stor_SetStateBits, Stor_Stat
};
// определяем объект для инициализации IStorage
IStorage _IStorage = { &IStorageMetods };
// пишем реализацию методов интерфейса
// для неиспользуемых методов ставим "заглушки" в виде E_NOTIMPL
HRESULT STDMETHODCALLTYPE Stor_QueryInterface ( IStorage FAR* Stor,
 REFIID riid, LPVOID FAR* ppvObj)
{
   return E_NOTIMPL; // не используется
}
HRESULT STDMETHODCALLTYPE Stor_AddRef ( IStorage FAR* Stor)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE Stor_Release ( IStorage FAR* Stor)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE Stor_CreateStream ( IStorage FAR* Stor,
 const WCHAR* pwcsName, DWORD grfMode, DWORD reserved1, DWORD reserved2,
 IStream** ppstm)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_OpenStream ( IStorage FAR* Stor,
 const WCHAR* pwcsName, void* reserved1, DWORD grfMode, DWORD reserved2,
 IStream** ppstm)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_CreateStorage ( IStorage FAR* Stor,
 const WCHAR* pwcsName, DWORD grfMode, DWORD reserved1, DWORD reserved2,
 IStorage** ppstg)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_OpenStorage ( IStorage FAR* Stor,
 const WCHAR* pwcsName, IStorage* pstgPriority, DWORD grfMode,
 SNB snbExclude, DWORD reserved, IStorage** ppstg)
```

```
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_CopyTo ( IStorage FAR* Stor,
 DWORD ciidExclude, IID const* rgiidExclude, SNB snbExclude,
 IStorage* pstgDest)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_MoveElementTo ( IStorage FAR* Stor,
 const OLECHAR* pwcsName, IStorage* pstgDest, const OLECHAR* pwcsNewName,
 DWORD grfFlags)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_Commit ( IStorage FAR* Stor,
 DWORD grfCommitFlags)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_Revert ( IStorage FAR* Stor)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_EnumElements (IStorage FAR* Stor,
 DWORD reserved1, void* reserved2, DWORD reserved3,
 IEnumSTATSTG** ppenum)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_DestroyElement ( IStorage FAR* Stor,
 const OLECHAR* pwcsName)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_RenameElement ( IStorage FAR* Stor,
 const WCHAR* pwcsOldName, const WCHAR* pwcsNewName)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_SetElementTimes ( IStorage FAR* Stor,
 const WCHAR* pwcsName, FILETIME const* pctime, FILETIME const* patime,
 FILETIME const* pmtime)
```

```
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_SetClass(IStorage FAR* This,
 REFCLSID clsid)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE Stor_SetStateBits ( IStorage FAR* Stor,
 DWORD grfStateBits, DWORD grfMask)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Stor_Stat ( IStorage FAR* Stor,
 STATSTG* pstatstg, DWORD grfStatFlag)
{
   return E_NOTIMPL;
}
```

Листинг 21.5. Реализация интерфейса `IOleClientSite`

```
// список методов интерфейса IOleClientSite
HRESULT STDMETHODCALLTYPE ClientSite_QueryInterface (
 IOleClientSite FAR* ClientSite, REFIID riid, void** ppvObject);
HRESULT STDMETHODCALLTYPE ClientSite_AddRef (
 IOleClientSite FAR* ClientSite);
HRESULT STDMETHODCALLTYPE ClientSite_Release (
 IOleClientSite FAR* ClientSite);
HRESULT STDMETHODCALLTYPE ClientSite_SaveObject(
 IOleClientSite FAR* ClientSite);
HRESULT STDMETHODCALLTYPE ClientSite_GetMoniker (
 IOleClientSite FAR* ClientSite, DWORD dwAssign, DWORD dwWhichMoniker,
 IMoniker** ppmk);
HRESULT STDMETHODCALLTYPE ClientSite_GetContainer (
 IOleClientSite FAR* ClientSite, LPOLECONTAINER FAR* ppContainer);
HRESULT STDMETHODCALLTYPE ClientSite_ShowObject (
 IOleClientSite FAR* ClientSite);
HRESULT STDMETHODCALLTYPE ClientSite_OnShowWindow (
 IOleClientSite FAR* ClientSite, BOOL fShow);
HRESULT STDMETHODCALLTYPE ClientSite_RequestNewObjectLayout (
 IOleClientSite FAR* ClientSite);
// определяем порядок методов для виртуальной таблицы IOleClientSite
IOleClientSiteVtbl IOleClientSiteMetods =
```

```
{
 ClientSite_QueryInterface, ClientSite_AddRef, ClientSite_Release,
 ClientSite_SaveObject, ClientSite_GetMoniker, ClientSite_GetContainer,
 ClientSite_ShowObject, ClientSite_OnShowWindow,
 ClientSite_RequestNewObjectLayout
};
// пишем реализацию методов интерфейса
HRESULT STDMETHODCALLTYPE ClientSite_AddRef (
 IOleClientSite FAR* ClientSite)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE ClientSite_Release (
 IOleClientSite FAR* ClientSite)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE ClientSite_SaveObject (
 IOleClientSite FAR* ClientSite)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE ClientSite_GetMoniker (
 IOleClientSite FAR* ClientSite, DWORD dwAssign, DWORD dwWhichMoniker,
 IMoniker** ppmk)
{
  return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE ClientSite_GetContainer (
 IOleClientSite FAR* ClientSite, LPOLECONTAINER FAR* ppContainer)
{
   // контейнер не поддерживается
   *ppContainer = 0;
   return E_NOINTERFACE; // указатель не возвращен
}
HRESULT STDMETHODCALLTYPE ClientSite_ShowObject (
IOleClientSite FAR* ClientSite)
{
   return NOERROR; // ошибок нет
}
HRESULT STDMETHODCALLTYPE ClientSite_OnShowWindow (
IOleClientSite FAR* ClientSite, BOOL fShow)
{
   return E_NOTIMPL;
}
```

```
HRESULT STDMETHODCALLTYPE ClientSite_RequestNewObjectLayout (
 IOleClientSite FAR* ClientSite)
{
   return E_NOTIMPL;
}
```

Листинг 21.6. Реализация интерфейса `IOleInPlaceSite`

```
// список методов интерфейса IOleInPlaceSite
HRESULT STDMETHODCALLTYPE InPlace_QueryInterface (
 IOleInPlaceSite FAR* Place, REFIID riid, void** ppvObject);
HRESULT STDMETHODCALLTYPE InPlace_AddRef ( IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_Release ( IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_GetWindow ( IOleInPlaceSite FAR* Place,
 HWND FAR* lphwnd);
HRESULT STDMETHODCALLTYPE InPlace_ContextSensitiveHelp (
 IOleInPlaceSite FAR* Place, BOOL fEnterMode);
HRESULT STDMETHODCALLTYPE InPlace_CanInPlaceActivate (
 IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_OnInPlaceActivate (
 IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_OnUIActivate (
 IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_GetWindowContext(
 IOleInPlaceSite FAR* Place, LPOLEINPLACEFRAME FAR* lplpFrame,
 LPOLEINPLACEUIWINDOW FAR* lplpDoc, LPRECT lprcPosRect,
 LPRECT lprcClipRect, LPOLEINPLACEFRAMEINFO lpFrameInfo);
HRESULT STDMETHODCALLTYPE InPlace_Scroll ( IOleInPlaceSite FAR* Place,
 SIZE scrollExtent);
HRESULT STDMETHODCALLTYPE InPlace_OnUIDeactivate (
 IOleInPlaceSite FAR*Place, BOOL fUndoable);
HRESULT STDMETHODCALLTYPE InPlace_OnInPlaceDeactivate (
 IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_DiscardUndoState (
 IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_DeactivateAndUndo (
 IOleInPlaceSite FAR* Place);
HRESULT STDMETHODCALLTYPE InPlace_OnPosRectChange (
IOleInPlaceSite FAR* Place, LPCRECT lprcPosRect);
// определяем порядок методов для виртуальной таблицы
IOleInPlaceSiteVtbl IOleInPlaceSiteMetods =
{
 InPlace_QueryInterface, InPlace_AddRef, InPlace_Release,
```

```
 InPlace_GetWindow, InPlace_ContextSensitiveHelp,
 InPlace_CanInPlaceActivate, InPlace_OnInPlaceActivate,
 InPlace_OnUIActivate, InPlace_GetWindowContext, InPlace_Scroll,
 InPlace_OnUIDeactivate, InPlace_OnInPlaceDeactivate,
 InPlace_DiscardUndoState, InPlace_DeactivateAndUndo,
 InPlace_OnPosRectChange
};
// пишем расширенную структуру
typedef struct
{
   IOleInPlaceSite PlaceSite; // наш объект для IOleInPlaceSite
   _IOleInPlaceFrame Frame; // наш объект для IOleInPlaceFrame
} _IOleInPlaceSite;
// пишем реализацию методов интерфейса
HRESULT STDMETHODCALLTYPE InPlace_AddRef ( IOleInPlaceSite FAR* Place)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE InPlace_Release ( IOleInPlaceSite FAR* Place)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE InPlace_GetWindow (
 IOleInPlaceSite FAR* Place, HWND FAR* lphwnd)
{
   // возвращаем указатель на окно браузера
   *lphwnd = ( ( _IOleInPlaceSite FAR*) Place)->Frame.BrowserWindow;
   // выходим
   return S_OK;
}
HRESULT STDMETHODCALLTYPE InPlace_ContextSensitiveHelp (
 IOleInPlaceSite FAR* Place, BOOL fEnterMode)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE InPlace_CanInPlaceActivate (
 IOleInPlaceSite FAR* Place)
{
       // Tell the browser we can in place activate
       return(S_OK);
}
HRESULT STDMETHODCALLTYPE InPlace_OnInPlaceActivate (
 IOleInPlaceSite FAR* Place)
```

```
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE InPlace_OnUIActivate (
 IOleInPlaceSite FAR* Place)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE InPlace_Scroll (
   IOleInPlaceSite FAR* Place, SIZE scrollExtent)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE InPlace_OnUIDeactivate (
 IOleInPlaceSite FAR* Place, BOOL fUndoable)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE InPlace_OnInPlaceDeactivate (
IOleInPlaceSite FAR* Place)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE InPlace_DiscardUndoState (
IOleInPlaceSite FAR* Place)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE InPlace_DeactivateAndUndo (
 IOleInPlaceSite FAR* Place)
{
   return E_NOTIMPL;
}
```

Листинг 21.7. Реализация интерфейса `IOleInPlaceFrame`

```
// список методов интерфейса IOleInPlaceFrame
HRESULT STDMETHODCALLTYPE Frame_QueryInterface (
 IOleInPlaceFrame FAR* Frame, REFIID riid, LPVOID FAR* ppvObj);
HRESULT STDMETHODCALLTYPE Frame_AddRef ( IOleInPlaceFrame FAR* Frame);
HRESULT STDMETHODCALLTYPE Frame_Release ( IOleInPlaceFrame FAR* Frame);
HRESULT STDMETHODCALLTYPE Frame_GetWindow ( IOleInPlaceFrame FAR* Frame,
 HWND FAR* lphwnd);
```

```
HRESULT STDMETHODCALLTYPE Frame_ContextSensitiveHelp (
 IOleInPlaceFrame FAR* Frame, BOOL fEnterMode);
HRESULT STDMETHODCALLTYPE Frame_GetBorder ( IOleInPlaceFrame FAR* Frame,
 LPRECT lprectBorder);
HRESULT STDMETHODCALLTYPE Frame_RequestBorderSpace (
 IOleInPlaceFrame FAR* Frame, LPCBORDERWIDTHS pborderwidths);
HRESULT STDMETHODCALLTYPE Frame_SetBorderSpace (
 IOleInPlaceFrame FAR* Frame, LPCBORDERWIDTHS pborderwidths);
HRESULT STDMETHODCALLTYPE Frame_SetActiveObject (
IOleInPlaceFrame FAR* Frame, IOleInPlaceActiveObject* pActiveObject,
 LPCOLESTR pszObjName);
HRESULT STDMETHODCALLTYPE Frame_InsertMenus (
 IOleInPlaceFrame FAR* Frame, HMENU hmenuShared,
 LPOLEMENUGROUPWIDTHS lpMenuWidths);
HRESULT STDMETHODCALLTYPE Frame_SetMenu ( IOleInPlaceFrame FAR* Frame,
 HMENU hmenuShared, HOLEMENU holemenu, HWND hwndActiveObject);
HRESULT STDMETHODCALLTYPE Frame_RemoveMenus (
 IOleInPlaceFrame FAR* Frame, HMENU hmenuShared);
HRESULT STDMETHODCALLTYPE Frame_SetStatusText (
 IOleInPlaceFrame FAR* Frame, LPCOLESTR pszStatusText);
HRESULT STDMETHODCALLTYPE Frame_EnableModeless (
 IOleInPlaceFrame FAR* Frame, BOOL fEnable);
HRESULT STDMETHODCALLTYPE Frame_TranslateAccelerator (
 IOleInPlaceFrame FAR* Frame, LPMSG lpmsg, WORD wID);
// определяем порядок методов для виртуальной таблицы
IOleInPlaceFrameVtbl IOleInPlaceFrameMetods =
{
 Frame_QueryInterface, Frame_AddRef, Frame_Release, Frame_GetWindow,
 Frame_ContextSensitiveHelp, Frame_GetBorder, Frame_RequestBorderSpace,
 Frame_SetBorderSpace, Frame_SetActiveObject, Frame_InsertMenus,
 Frame_SetMenu, Frame_RemoveMenus, Frame_SetStatusText,
 Frame_EnableModeless, Frame_TranslateAccelerator
};
// пишем расширенную структуру
typedef struct
{
   IOleInPlaceFrame Frame; // первый объект браузера
   HWND BrowserWindow; // дескриптор окна браузера
} _IOleInPlaceFrame;
// пишем реализацию методов интерфейса
HRESULT STDMETHODCALLTYPE Frame_QueryInterface (
 IOleInPlaceFrame FAR* Frame, REFIID riid, LPVOID FAR* ppvObj)
{
   return E_NOTIMPL;
}
```

```
HRESULT STDMETHODCALLTYPE Frame_AddRef ( IOleInPlaceFrame FAR* Frame)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE Frame_Release ( IOleInPlaceFrame FAR* Frame)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE Frame_GetWindow ( IOleInPlaceFrame FAR* Frame,
 HWND FAR* lphwnd)
{
   // возвращаем указатель на окно браузера
   *lphwnd = ( ( _IOleInPlaceFrame*) Frame)->BrowserWindow;
   return S_OK; // выходим
}
HRESULT STDMETHODCALLTYPE Frame_ContextSensitiveHelp (
 IOleInPlaceFrame FAR* Frame, BOOL fEnterMode)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Frame_GetBorder ( IOleInPlaceFrame FAR* Frame,
 LPRECT lprectBorder)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Frame_RequestBorderSpace (
 IOleInPlaceFrame FAR* Frame, LPCBORDERWIDTHS pborderwidths)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Frame_SetBorderSpace (
 IOleInPlaceFrame FAR* Frame, LPCBORDERWIDTHS pborderwidths)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Frame_SetActiveObject (
 IOleInPlaceFrame FAR* Frame, IOleInPlaceActiveObject *pActiveObject,
 LPCOLESTR pszObjName)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE Frame_InsertMenus (
 IOleInPlaceFrame FAR* Frame, HMENU hmenuShared,
 LPOLEMENUGROUPWIDTHS lpMenuWidths)
```

```
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Frame_SetMenu ( IOleInPlaceFrame FAR* Frame,
HMENU hmenuShared, HOLEMENU holemenu, HWND hwndActiveObject)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE Frame_RemoveMenus (
 IOleInPlaceFrame FAR* Frame, HMENU hmenuShared)
{
   return E_NOTIMPL;
}
HRESULT STDMETHODCALLTYPE Frame_SetStatusText (
 IOleInPlaceFrame FAR* Frame, LPCOLESTR pszStatusText)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE Frame_EnableModeless (
 IOleInPlaceFrame FAR* Frame, BOOL fEnable)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE Frame_TranslateAccelerator (
 IOleInPlaceFrame FAR* Frame, LPMSG lpmsg, WORD wID)
{
   return E_NOTIMPL;
}
```

Листинг 21.8. Реализация интерфейса `IDocHostUIHandler`

```
// список методов интерфейса IDocHostUIHandler
HRESULT STDMETHODCALLTYPE UI_QueryInterface (
 IDocHostUIHandler FAR* UIHandler, REFIID riid, void** ppvObject);
HRESULT STDMETHODCALLTYPE UI_AddRef ( IDocHostUIHandler FAR* UIHandler);
HRESULT STDMETHODCALLTYPE UI_Release (
 IDocHostUIHandler FAR* UIHandler);
HRESULT STDMETHODCALLTYPE UI_ShowContextMenu (
 IDocHostUIHandler FAR* UIHandler, DWORD dwID, POINT __RPC_FAR*ppt,
 IUnknown __RPC_FAR* pcmdtReserved, IDispatch __RPC_FAR* pdispReserved);
HRESULT STDMETHODCALLTYPE UI_GetHostInfo (
 IDocHostUIHandler FAR* UIHandler, DOCHOSTUIINFO __RPC_FAR* pInfo);
```

```
HRESULT STDMETHODCALLTYPE UI_ShowUI ( IDocHostUIHandler FAR* UIHandler,
 DWORD dwID, IOleInPlaceActiveObject __RPC_FAR* pActiveObject,
 IOleCommandTarget __RPC_FAR* pCommandTarget,
 IOleInPlaceFrame __RPC_FAR* pFrame,
 IOleInPlaceUIWindow __RPC_FAR* pDoc);
HRESULT STDMETHODCALLTYPE UI_HideUI ( IDocHostUIHandler FAR* UIHandler);
HRESULT STDMETHODCALLTYPE UI_UpdateUI (
 IDocHostUIHandler FAR* UIHandler);
HRESULT STDMETHODCALLTYPE UI_EnableModeless (
 IDocHostUIHandler FAR* UIHandler, BOOL fEnable);
HRESULT STDMETHODCALLTYPE UI_OnDocWindowActivate (
 IDocHostUIHandler FAR* UIHandler, BOOL fActivate);
HRESULT STDMETHODCALLTYPE UI_OnFrameWindowActivate (
 IDocHostUIHandler FAR* UIHandler, BOOL fActivate);
HRESULT STDMETHODCALLTYPE UI_ResizeBorder (
 IDocHostUIHandler FAR* UIHandler, LPCRECT prcBorder,
 IOleInPlaceUIWindow __RPC_FAR* pUIWindow, BOOL fRameWindow);
HRESULT STDMETHODCALLTYPE UI_TranslateAccelerator (
 IDocHostUIHandler FAR* UIHandler, LPMSG lpMsg,
 const GUID __RPC_FAR* pguidCmdGroup, DWORD nCmdID);
HRESULT STDMETHODCALLTYPE UI_GetOptionKeyPath (
 IDocHostUIHandler FAR* UIHandler, LPOLESTR __RPC_FAR* pchKey,
 DWORD dw);
HRESULT STDMETHODCALLTYPE UI_GetDropTarget (
 IDocHostUIHandler FAR* UIHandler, IDropTarget __RPC_FAR* pDropTarget,
 IDropTarget __RPC_FAR* __RPC_FAR* ppDropTarget);
HRESULT STDMETHODCALLTYPE UI_GetExternal (
 IDocHostUIHandler FAR* UIHandler,
 IDispatch __RPC_FAR* __RPC_FAR* ppDispatch);
HRESULT STDMETHODCALLTYPE UI_TranslateUrl (
 IDocHostUIHandler FAR* UIHandler, DWORD dwTranslate,
 OLECHAR __RPC_FAR* pchURLIn, OLECHAR __RPC_FAR*__RPC_FAR* ppchURLOut);
HRESULT STDMETHODCALLTYPE UI_FilterDataObject (
 IDocHostUIHandler FAR* UIHandler, IDataObject __RPC_FAR* pDO,
 IDataObject __RPC_FAR* __RPC_FAR* ppDORet);
// определяем порядок методов для виртуальной таблицы
IDocHostUIHandlerVtbl IDocHostUIHandlerMetods =
{
 UI_QueryInterface, UI_AddRef, UI_Release, UI_ShowContextMenu,
 UI_GetHostInfo, UI_ShowUI, UI_HideUI, UI_UpdateUI, UI_EnableModeless,
 UI_OnDocWindowActivate, UI_OnFrameWindowActivate, UI_ResizeBorder,
 UI_TranslateAccelerator, UI_GetOptionKeyPath, UI_GetDropTarget,
 UI_GetExternal, UI_TranslateUrl, UI_FilterDataObject
};
```

```
// пишем реализацию методов интерфейса
HRESULT STDMETHODCALLTYPE UI_AddRef ( IDocHostUIHandler FAR* UIHandler)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE UI_Release ( IDocHostUIHandler FAR* UIHandler)
{
   return 1;
}
HRESULT STDMETHODCALLTYPE UI_ShowContextMenu (
 IDocHostUIHandler FAR* UIHandler, DWORD dwID, POINT __RPC_FAR* ppt,
 IUnknown __RPC_FAR* pcmdtReserved, IDispatch __RPC_FAR* pdispReserved)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_GetHostInfo (
 IDocHostUIHandler FAR* UIHandler, DOCHOSTUIINFO __RPC_FAR* pInfo)
{
   // инициализация и установка внешнего вида окна просмотра
   pInfo->cbSize = sizeof ( DOCHOSTUIINFO);
   // устанавливаем плоскую линейку прокрутки и рамку окна
   pInfo->dwFlags = DOCHOSTUIFLAG_FLAT_SCROLLBAR |
                    DOCHOSTUIFLAG_FLAT_SCROLLBAR;
   // устанавливаем реакцию на двойной щелчок мыши
   pInfo->dwDoubleClick = DOCHOSTUIDBLCLK_DEFAULT;
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_ShowUI ( IDocHostUIHandler FAR* UIHandler,
 DWORD dwID, IOleInPlaceActiveObject __RPC_FAR* pActiveObject,
 IOleCommandTarget __RPC_FAR* pCommandTarget,
 IOleInPlaceFrame __RPC_FAR* pFrame,
 IOleInPlaceUIWindow __RPC_FAR* pDoc)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_HideUI ( IDocHostUIHandler FAR* UIHandler)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_UpdateUI (
 IDocHostUIHandler FAR* UIHandler)
{
   return S_OK;
}
```

```
HRESULT STDMETHODCALLTYPE UI_EnableModeless (
 IDocHostUIHandler FAR* UIHandler, BOOL fEnable)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_OnDocWindowActivate (
 IDocHostUIHandler FAR* UIHandler, BOOL fActivate)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_OnFrameWindowActivate (
 IDocHostUIHandler FAR* UIHandler, BOOL fActivate)
{
   return S_OK;
}
HRESULT STDMETHODCALLTYPE UI_ResizeBorder (
 IDocHostUIHandler FAR* UIHandler, LPCRECT prcBorder,
 IOleInPlaceUIWindow __RPC_FAR* pUIWindow, BOOL fRameWindow)
{
   return S_OK;
)
HRESULT STDMETHODCALLTYPE UI_TranslateAccelerator (
 IDocHostUIHandler FAR* UIHandler, LPMSG lpMsg,
 const GUID __RPC_FAR* pguidCmdGroup, DWORD nCmdID)
{
   return S_FALSE;
}
HRESULT STDMETHODCALLTYPE UI_GetOptionKeyPath (
 IDocHostUIHandler FAR* UIHandler, LPOLESTR __RPC_FAR* pchKey, DWORD dw)
{
   return S_FALSE;
}
HRESULT STDMETHODCALLTYPE UI_GetDropTarget (
 IDocHostUIHandler FAR* UIHandler, IDropTarget __RPC_FAR* pDropTarget,
 IDropTarget __RPC_FAR*__RPC_FAR* ppDropTarget)
{
   return S_FALSE;
}
HRESULT STDMETHODCALLTYPE UI_GetExternal (
 IDocHostUIHandler FAR* UIHandler,
 IDispatch __RPC_FAR* __RPC_FAR* ppDispatch)
{
   *ppDispatch = 0;
```

```
   return S_FALSE;
}
HRESULT STDMETHODCALLTYPE UI_TranslateUrl (
 IDocHostUIHandler FAR* UIHandler, DWORD dwTranslate,
 OLECHAR __RPC_FAR* pchURLIn, OLECHAR __RPC_FAR* __RPC_FAR* ppchURLOut)
{
   *ppchURLOut = 0;
   return S_FALSE;
}
HRESULT STDMETHODCALLTYPE UI_FilterDataObject (
 IDocHostUIHandler FAR* UIHandler, IDataObject __RPC_FAR* pDO,
 IDataObject __RPC_FAR* __RPC_FAR* ppDORet)
{
   *ppDORet = 0;
   return S_FALSE;
}
```

Как видите, реализация интерфейсов занимает много времени, но это необходимое условие их использования. Вначале мы определили все методы интерфейса (в том числе и базового `IUnknown`). После этого составили виртуальную таблицу указателей на методы. Виртуальная таблица представляет собой массив указателей на методы интерфейса и позволяет динамически связать абстрактный метод с его реализацией. В завершении выполнили реализацию всех методов интерфейса. Не задействованные в программе методы оформили в виде "заглушек", просто вернув значение `E_NOTIMPL`. Описание всех методов можно найти в документации фирмы Microsoft (например, на сайте). Для непосредственного управления возможностями браузера дополнительно понадобится еще один интерфейс `IWebBrowser2`, указатель на который мы получим по ходу программы.

Теперь следует создать в редакторе пустой файл WebBrowser.c, а в редакторе ресурсов — меню. Добавьте в меню пункт **File** с подпунктами **Open URL** и **Exit** и пункт **Browse** с подпунктами **Refresh**, **Go Back**, **Go Forward**, **Stop**, **Home** и **Search**. В дальнейшем идентификаторы подпунктов меню будут применяться для вызова тех или иных функций.

На этом все основные операции закончены, и можно приступать к написанию основного кода программы. В листинге 21.9 показано, как это нужно сделать.

Листинг 21.9. Основной код программы браузера в файле WebBrowser.c

```
// макрос для подключения функции CoInitializeEx
#define _WIN32_DCOM
// подключаем необходимые файлы
#include <windows.h> // общие функции Win32
```

```
#include <stdio.h> // работа с файлами
#include <exdisp.h> // интерфейс IWebBrowser2
#include <mshtmhst.h> // интерфейс IDocHostUIHandler
// подключаем интерфейсы
#include "IStorage.h"
#include "IOleInPlaceFrame.h"
#include "IOleClientSite.h"
#include "IDocHostUIHandler.h"
#include "IOleInPlaceSite.h"
// определяем глобальные переменные
LPCTSTR lpszClassName = "BrowserClass"; // имя класса окна
// общая структура поддержки интерфейсов
typedef struct
{
   IOleClientSite ClientSite;
   _IOleInPlaceSite PlaceSite;
   IDocHostUIHandler UI_handler;
} _IOleClientSite;
// определяем функции
LRESULT CALLBACK WindowMainProc ( HWND hMainWnd, UINT message,
 WPARAM wParam,
LPARAM lParam); // главная функция окна
int AttachWindow ( HWND hWnd); // присоединение окна браузера
void DetachWindow ( HWND hWnd); // удаление окна браузера
void ReleaseBrowser ( HWND hMainWnd); // завершение работы браузера
void ShowPage (HWND hWnd, LPTSTR lpURL); // загрузка веб-страницы
// изменение размеров окна браузера
void SetSizeWindow ( HWND hWnd, long width, long height);
void Refresh ( HWND hWnd); // обновление страницы
void GoBack ( HWND hWnd); // вернуться к предыдущей странице
void GoForward ( HWND hWnd); // перейти к следующей странице
void Stop ( HWND hWnd); // остановить просмотр страницы
void Search ( HWND hWnd); // загрузить страницу поиска по умолчанию
void Home ( HWND hWnd); // перейти на домашнюю страницу
// основная функция программы
int CALLBACK WinMain ( HINSTANCE hInstance, HINSTANCE hPrewInstance,
 LPSTR lpCmdLine, int nCmdShow)
{
   MSG msg; // поддержка сообщений
   HWND hWnd = NULL; // дескриптор окна
   WNDCLASSEX wce; // расширенная структура окна
   // инициализируем объектную модель COM
   if ( CoInitializeEx ( 0, COINIT_APARTMENTTHREADED |
                              COINIT_SPEED_OVER_MEMORY) != S_OK) return -1;
```

```
    // обнуляем структуру окна
    ZeroMemory ( &wce, sizeof ( WNDCLASSEX));
    // определяем начальные свойства окна
    wce.cbSize = sizeof ( WNDCLASSEX); // размер структуры
    wce.hInstance = hInstance; // дескриптор программы
    wce.style = CS_HREDRAW | CS_VREDRAW; // стиль окна
    wce.lpfnWndProc = WindowMainProc; // основная функция окна
    wce.lpszClassName = lpszClassName; // имя для класса окна
    // дескриптор меню, созданного в редакторе ресурсов
    wce.lpszMenuName = MAKEINTRESOURCE ( IDR_MYMENU);
    // регистрируем созданный класс окна в системе
    RegisterClassEx ( &wce);
    // создаем основное окно программы
    hWnd = CreateWindowEx ( 0, lpszClassName, "My Browser",
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL,
        NULL, hInstance, 0);
    // выводим созданное окно на экран монитора
    ShowWindow(hWnd, nCmdShow);
    // и перерисовываем его для удаления нежелательных эффектов
    UpdateWindow ( hWnd);
    // запускаем цикл обработки сообщений
    while ( GetMessage ( &msg, 0, 0, 0))
    {
      TranslateMessage ( &msg);
      DispatchMessage ( &msg);
    }
    // освобождаем ресурсы COM
    CoUninitialize ();
    return 0;
}
// главная оконная функция
LRESULT CALLBACK WindowMainProc ( HWND hMainWnd, UINT message,
 WPARAM wParam, LPARAM lParam)
{
    // обрабатываем сообщения
    switch ( message)
    {
     case WM_SIZE: // обработка размеров окна
      {
        // изменяем размеры окна просмотра веб-страниц
        SetSizeWindow ( hMainWnd, LOWORD ( lParam), HIWORD ( lParam));
      }
      return 0;
```

```
case WM_CREATE: // создание окна
 {
   // подключаем окно браузера
   if ( AttachWindow ( hMainWnd)) return -1;
   // пытаемся загрузить страницу по умолчанию
   ShowPage ( hMainWnd, http://www.yandex.ru);
 }
 return 0;
case WM_SYSCOMMAND: // обработка системного меню
 {
  switch ( wParam & 0xFFF0)
   {
    case SC_CLOSE: // выбрана кнопка закрытия окна или Alt + F4
      ReleaseBrowser ( hMainWnd); // закрываем браузер
      return TRUE;
    default: // обработка по умолчанию
      break;
   }
 }
 break;
case WM_COMMAND: // обработка команд меню
 {
  switch ( LOWORD ( wParam))
   {
    case IDM_OPEN: // открытие файлов
     {
       // определяем маску для файлов
       char szFilters[] = "HTML Files (url, htm, \
                              html)\0*.url;*.htm;*.html\0\0";
       char szUrl[MAX_PATH] = " "; // путь
       OPENFILENAME ofn;
       // обнуляем и заполняем структуру данными
       ZeroMemory ( &ofn, sizeof ( OPENFILENAME));
       ofn.lStructSize = sizeof ( OPENFILENAME);
       ofn.hwndOwner = hMainWnd;
       ofn.lpstrFilter = szFilters;
       ofn.lpstrFile = szUrl;
       ofn.nMaxFile = sizeof ( szUrl);
       ofn.Flags = OFN_HIDEREADONLY | OFN_FILEMUSTEXIST;
       // выводим на экран диалог для открытия файлов
       if( GetOpenFileName ( &ofn))
       {
         int ch = 0; // код символа
         FILE* f = NULL; // дескриптор файла
```

```
          // открываем файл для чтения
          f = fopen ( ofn.lpstrFile, "rb");
          // обрабатываем файл
          while ( !feof ( f))
          {
            ch = getc ( f); // читаем символ
            // если это двоеточие, пытаемся прочитать адрес
            if ( ( char) ch == ':')
            {
              char tmp[100]; // временный буфер
              int i = 0; // счетчик
              // копируем начало адреса
              strcpy ( szUrl, "http:");
              // выделяем из строки адрес страницы
              for ( i; i < 100; i++)
              {
                tmp[i] = getc ( f);
                if ( isspace ( ( int) tmp[i]))
                {
                  tmp[i] = '\0';
                  strcat ( szUrl, tmp);
                  goto end; // выходим из циклов
                }
              }
            }
          }
end:
          if ( f) fclose ( f); // закрываем файл
          // загружаем веб-страницу в браузер
        }
      }
      break;
    case IDM_EXIT: // выбран пункт меню Exit
      ReleaseBrowser ( hMainWnd);
      return TRUE;
    case IDM_BACK: // выбран пункт меню Go Back
      GoBack ( hMainWnd);
      break;
    case IDM_FORWARD: // выбран пункт меню Go Forward
      GoForward ( hMainWnd);
      break;
    case IDM_STOP: // выбран пункт меню Stop
      Stop ( hMainWnd);
      break;
```

```
        case IDM_REFRESH: // выбран пункт меню Refresh
           Refresh ( hMainWnd);
           break;
        case IDM_HOME: // выбран пункт меню Home
        {
           Home ( hMainWnd);
        }
        break;
        case IDM_SEARCH: // выбран пункт меню Search
           Search ( hMainWnd);
           break;
       }
     }
     break;
    case WM_DESTROY: // завершение работы программы
     {
       ReleaseBrowser ( hMainWnd); // закрываем браузер
     }
     return TRUE;
   }
   // обработка сообщений по умолчанию
   return DefWindowProc ( hMainWnd, message, wParam, lParam);
}
// функция закрытия окна браузера
void ReleaseBrowser ( HWND hMainWnd)
{
   // удаляем окна браузера
   DetachWindow ( hMainWnd);
   // завершаем работу программы
   PostQuitMessage ( 0);
}
// присоединение окна браузера к главному окну программы
int AttachWindow ( HWND hWnd)
{
   // объявляем указатели на интерфейсы
   IOleObject* Browser; // объект для браузера
   IWebBrowser2* WebBrowser2; // интерфейс поддержки браузера
   _IOleClientSite* Site; // внешний вид, свойства и сообщения браузера
   RECT r; // управление размером окна браузера
   char* p = NULL; // указатель
   // выделяем необходимый размер памяти для объекта браузера
   if ( !( p = ( char*) GlobalAlloc (GMEM_FIXED,
     sizeof ( IOleObject*) + sizeof ( _IOleClientSite))) return 1;
```

```
  // инициализируем объект и определяем виртуальные таблицы методов
  Site = ( _IOleClientSite*) ( sizeof ( IOleObject*) + p);
  Site->ClientSite.lpVtbl = &IOleClientSiteMetods;
  Site->PlaceSite.Frame.Frame.lpVtbl = &IOleInPlaceFrameMetods;
  Site->PlaceSite.PlaceSite.lpVtbl = &IOleInPlaceSiteMetods;
  Site->UI_handler.lpVtbl = &IDocHostUIHandlerMetods;
  // копируем дескриптор главного окна программы
  Site->PlaceSite.Frame.BrowserWindow = hWnd;
  // создаем объект браузера
  if ( !OleCreate ( &CLSID_WebBrowser, &IID_IOleObject, OLERENDER_DRAW,
        0, ( IOleClientSite*) Site, &_IStorage, ( void**) &Browser))
  {
    // присваиваем указатель на полученный объект браузера
    *( ( IOleObject**) p) = Browser;
    // назначаем имя для полученного объекта
    Browser->lpVtbl->SetHostNames ( Browser, L"Browser Host", 0);
    // сохраняем указатель для главного окна программы
    SetWindowLong ( hWnd, GWL_USERDATA, ( LONG) p);
    // получаем размеры клиентской части главного окна
    GetClientRect ( hWnd, &r);
    // сообщаем объекту браузера о внедрении его в наш контейнер OLE
    if ( !OleSetContainedObject ( ( struct IUnknown*) Browser, TRUE)
       && !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
       ( void**) &WebBrowser2) && !Browser->lpVtbl->DoVerb ( Browser,
       OLEIVERB_UIACTIVATE | OLEIVERB_SHOW, NULL,
       ( IOleClientSite*) Site, -1, hWnd, &r))
    {
      // устанавливаем начальные размеры окна браузера
      WebBrowser2->lpVtbl->put_Left ( WebBrowser2, 0);
      WebBrowser2->lpVtbl->put_Top ( WebBrowser2, 0);
      WebBrowser2->lpVtbl->put_Width ( WebBrowser2, r.right);
      WebBrowser2->lpVtbl->put_Height ( WebBrowser2, r.bottom);
      // уменьшаем счетчик ссылок на объект
      WebBrowser2->lpVtbl->Release ( WebBrowser2);
      // выходим из функции
      return 0;
    }
  }
  // освобождаем память
  GlobalFree ( p);
  // выходим с ошибкой
  return 1;
}
```

```
// освобождаем окно браузера
void DetachWindow ( HWND hWnd)
{
   IOleObject* Browser = NULL, **TmpBrowser = NULL;
   // получаем указатель на объект браузера
   if ( ( TmpBrowser = ( IOleObject**) GetWindowLong ( hWnd,\
                      GWL_USERDATA)))
   {
     Browser = *TmpBrowser;
     // останавливаем выполнение объекта и удаляем его
     Browser->lpVtbl->Close ( Browser, OLECLOSE_NOSAVE);
     // уменьшаем счетчик ссылок на объект
     Browser->lpVtbl->Release ( Browser);
     // освобождаем выделенную ранее память
     GlobalFree ( TmpBrowser);
   }
}
// получаем указатель на интерфейс IDocHostUIHandler
HRESULT STDMETHODCALLTYPE UI_QueryInterface (
 IDocHostUIHandler FAR* UIHandler, REFIID riid, LPVOID FAR* ppvObj)
{
   return ( ClientSite_QueryInterface ( ( IOleClientSite*)
          ( ( char*) UIHandler - sizeof ( IOleClientSite)
          sizeof ( _IOleInPlaceSite)), riid, ppvObj));
}
// получаем указатель на интерфейс IOleClientSite
HRESULT STDMETHODCALLTYPE ClientSite_QueryInterface (
 IOleClientSite FAR* ClientSite, REFIID riid, void** ppvObject)
{
   if ( !memcmp ( riid, &IID_IUnknown, sizeof ( GUID)) ||
        !memcmp ( riid, &IID_IOleClientSite, sizeof ( GUID)))
     *ppvObject = &( ( _IOleClientSite*) ClientSite)->ClientSite;
   else if ( !memcmp ( riid, &IID_IDocHostUIHandler, sizeof ( GUID)))
     *ppvObject = &( ( _IOleClientSite*) ClientSite)->UI_handler;
   else if ( !memcmp ( riid, &IID_IOleInPlaceSite, sizeof ( GUID)))
      *ppvObject = &( ( _IOleClientSite*) ClientSite)->PlaceSite;
   else
   {
     *ppvObject = 0;
     return E_NOINTERFACE; // не удалось получить указатель
   }
   return S_OK; // указатель на интерфейс успешно получен
}
```

```
// получаем указатель на интерфейс IOleInPlaceSite
HRESULT STDMETHODCALLTYPE InPlace_QueryInterface (
 IOleInPlaceSite FAR* Frame, REFIID riid, LPVOID FAR* ppvObj)
{
   return ( ClientSite_QueryInterface ( ( IOleClientSite*)
          ( ( char*) Frame - sizeof ( IOleClientSite)), riid,
          ppvObj));
}
// определяем параметры окна для объекта браузера
HRESULT STDMETHODCALLTYPE InPlace_GetWindowContext (
 IOleInPlaceSite FAR* PlaceSite, LPOLEINPLACEFRAME FAR* lplpFrame,
 LPOLEINPLACEUIWINDOW FAR* lplpDoc, LPRECT lprcPosRect,
 LPRECT lprcClipRect, LPOLEINPLACEFRAMEINFO lpFrameInfo)
{
   *lplpFrame = ( LPOLEINPLACEFRAME) & ( (
                  _IOleInPlaceSite*) PlaceSite)->Frame;
   *lplpDoc = 0; // то же самое окно для документа
   // заполняем информационную структуру для окна контейнера
   lpFrameInfo->fMDIApp = FALSE; // не MDI-приложение
   // дескриптор окна контейнера верхнего уровня
   lpFrameInfo->hwndFrame = ( ( _IOleInPlaceFrame*)
                              *lplpFrame)->BrowserWindow;
   lpFrameInfo->haccel = 0; // дескриптор таблицы акселераторов
   lpFrameInfo->cAccelEntries = 0; // число таблиц
   return S_OK;
}
// функция для загрузки веб-страницы
void ShowPage ( HWND hWnd, LPTSTR lpURL)
{
   IOleObject* Browser;
   IWebBrowser2* WebBrowser2;
   VARIANT URL;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
   // подключаем интерфейс поддержки браузера
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                            ( void**) &WebBrowser2))
   {
     // инициализируем вариантный тип для указателя на адрес
     VariantInit ( &URL);
     // определяем тип переменной как BSTR
     URL.vt = VT_BSTR;
     #ifndef UNICODE // если используется Unicod
```

```
      {
        wchar_t* w_tmp;
        DWORD dwWCharSyze;
        // определяем размер для символьного буфера
        dwWCharSyze = MultiByteToWideChar ( CP_ACP, 0, lpURL, -1, 0, 0);
        // пытаемся выделить необходимое количество памяти
        if ( !( w_tmp = ( wchar_t*) GlobalAlloc ( GMEM_FIXED,
                 sizeof ( wchar_t)* dwWCharSyze)))
             goto no_memory; // не удалось выделить память
        // преобразуем в строку символов ANSI
        MultiByteToWideChar ( CP_ACP, 0, lpURL, -1, w_tmp, dwWCharSyze);
        // копируем строку из буфера
        URL.bstrVal = SysAllocString ( w_tmp);
        // освобождаем память
        GlobalFree ( w_tmp);
      }
      #else
        // копируем строку без преобразования
        URL.bstrVal = SysAllocString ( ( const OLECHAR*) lpURL);
      #endif
      // если данные отсутствуют
      if ( !URL.bstrVal)
      {
        // освобождаем ссылку на интерфейс
no_memory:
        WebBrowser2->lpVtbl->Release ( WebBrowser2);
        return;
      }
      // загружаем указанную веб-страницу
      WebBrowser2->lpVtbl->Navigate2 ( WebBrowser2, &URL, 0, 0, 0, 0);
      // освобождаем ресурсы
      VariantClear ( &URL);
      // освобождаем ссылку на интерфейс
      WebBrowser2->lpVtbl->Release ( WebBrowser2);
    }
}
// функция для перехода к предыдущей странице
void GoBack ( HWND hWnd)
{
   IWebBrowser2* WebBrowser2;
   IOleObject* Browser;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
```

```
   // получаем указатель на интерфейс IWebBrowser2
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                          ( void**) &WebBrowser2))
   {
     // переходим к предыдущей странице
     WebBrowser2->lpVtbl->GoBack ( WebBrowser2);
     // освобождаем ссылку на интерфейс
     WebBrowser2->lpVtbl->Release ( WebBrowser2);
   }
}
// функция для перехода к следующей странице
void GoForward ( HWND hWnd)
{
   IWebBrowser2* WebBrowser2;
   IOleObject* Browser;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
   // получаем указатель на интерфейс IWebBrowser2
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                          ( void**) &WebBrowser2))
   {
     // переходим к следующей странице
     WebBrowser2->lpVtbl->GoForward ( WebBrowser2);
     // освобождаем ссылку на интерфейс
     WebBrowser2->lpVtbl->Release ( WebBrowser2);
   }
}
// функция для остановки загружаемой страницы
void Stop ( HWND hWnd)
{
   IWebBrowser2* WebBrowser2;
   IOleObject* Browser;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
   // получаем указатель на интерфейс IWebBrowser2
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                          ( void**) &WebBrowser2))
   {
     // останавливаем загрузку страницы
     WebBrowser2->lpVtbl->Stop ( WebBrowser2);
     // освобождаем ссылку на интерфейс
     WebBrowser2->lpVtbl->Release ( WebBrowser2);
   }
}
```

```
// функция для перехода к домашней странице
void Home ( HWND hWnd)
{
   IWebBrowser2* WebBrowser2;
   IOleObject* Browser;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
   // получаем указатель на интерфейс IWebBrowser2
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                           ( void**) &WebBrowser2))
   {
     // переходим к домашней странице
     WebBrowser2->lpVtbl->GoHome ( WebBrowser2);
     // освобождаем ссылку на интерфейс
     WebBrowser2->lpVtbl->Release ( WebBrowser2);
   }
}
// функция для перехода к странице поиска
void Search ( HWND hWnd)
{
   IWebBrowser2* WebBrowser2;
   IOleObject* Browser;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
   // получаем указатель на интерфейс IWebBrowser2
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                           ( void**) &WebBrowser2))
   {
     // переходим к поисковой странице
     WebBrowser2->lpVtbl->GoSearch ( WebBrowser2);
     // освобождаем ссылку на интерфейс
     WebBrowser2->lpVtbl->Release ( WebBrowser2);
   }
}
// функция для обновления текущей страницы
void Refresh ( HWND hWnd)
{
   IWebBrowser2* WebBrowser2;
   IOleObject* Browser;
   // получаем указатель на объект браузера
   Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
   // получаем указатель на интерфейс IWebBrowser2
   if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                           ( void**) &WebBrowser2))
```

```
    {
      // обновляем текущую страницу
      WebBrowser2->lpVtbl-> Refresh ( WebBrowser2);
      // освобождаем ссылку на интерфейс
      WebBrowser2->lpVtbl->Release ( WebBrowser2);
    }
}
// функция для обработки размеров окна браузера
void SetSizeWindow ( HWND hWnd, long width, long height)
{
    IWebBrowser2* WebBrowser2;
    IOleObject* Browser;
    // получаем указатель на объект браузера
    Browser = *( ( IOleObject**) GetWindowLong ( hWnd, GWL_USERDATA));
    // получаем указатель на интерфейс IWebBrowser2
    if ( !Browser->lpVtbl->QueryInterface ( Browser, &IID_IWebBrowser2,
                                            ( void**) &WebBrowser2))
    {
      // устанавливаем новые размеры окна
      WebBrowser2->lpVtbl->put_Width ( WebBrowser2, width); // ширина
      WebBrowser2->lpVtbl->put_Height ( WebBrowser2, height); // высота
      // освобождаем ссылку на интерфейс
      WebBrowser2->lpVtbl->Release ( WebBrowser2);
    }
}
```

Скомпилируйте файл программы и оцените полученный результат. Если в момент запуска приложения связь с провайдером отсутствует, будет загружен стандартный диалог дозвонки. С помощью меню **Open URL** можно загружать файлы с расширениями htm, html и url (ссылки на сайты из папки **Избранное**). Сразу хочу заметить, что алгоритм поиска адреса в файле очень примитивен и представлен только ради демонстрации одного из способов загрузки веб-страниц. В коммерческих программах потребуется написать более сложный анализатор файлов для получения корректной адресной строки.

При желании можете самостоятельно добавить дополнительные возможности, поддерживаемые интерфейсом `IWebBrowser2`, а также обработку меню и сообщений внедренного объекта браузера.

21.3. Загрузка файлов

Важной составляющей современных браузеров является возможность сохранения файлов из сети на компьютер пользователя. Однако реализация этой задачи в некоторых программах оставляет желать лучшего, и разработчикам

приходится решать ее самостоятельно. Поскольку эта тема интересует многих, я постараюсь познакомить читателей с основами загрузки различных типов файлов из глобальной сети.

Во-первых, условно разделим все потенциальные источники файлов на две части: поддерживающие протокол HTTP и протокол FTP. Существуют и другие протоколы передачи данных, но они нас в данном случае не интересуют. Для каждого из указанных протоколов имеется набор сервисных и общих функций, позволяющих легко применять их на практике, не задумываясь о тонкостях внутреннего строения и способов организации обмена данными. Чтобы подключить к программе функции Интернета, необходимо добавить в проект файл определений Wininet.h, а в опциях компоновщика установить ссылку на библиотеку Wininet.lib.

Перед началом работы рекомендуется всегда вызывать функцию `InternetGetConnectedState`, чтобы проверить текущее состояние подключения к сети. Если соединение активно, можно начинать инициализацию посредством функции `InternetOpen`, в противном случае с помощью `InternetAttemptConnect` следует активизировать диалоговое окно для установки соединения с сетью. После этого можно открывать требуемый сетевой ресурс и начинать чтение файла. Чтобы упростить использование функций Интернета, напишем отдельный класс `DownloadFile`. Пример реализации такого класса показан в листингах 21.10 и 21.11.

Листинг 21.10. Файл DownloadFile.h класса `DownloadFile`

```
// если вы работаете не с VC++.NET, можно определить константы вручную
#define INTERNET_RAS_INSTALLED              0x10
#define INTERNET_CONNECTION_OFFLINE         0x20
#define INTERNET_CONNECTION_CONFIGURED      0x40
// файл определений для функций Интернета
#include <wininet.h>
// объявляем класс
class DownloadFile
{
public:
   DownloadFile (); // конструктор
   ~DownloadFile (); // деструктор
// открытые функции класса
   bool Initialize ( bool bConnect = false); // инициализация
   // загрузка файла с HTTP
   int GetFile ( const char* address, const char* out_file);
   // загрузка файла с FTP-сервера без информации о текущем состоянии
   int GetFTPFile ( const char* ftp_address, const char* in_file,
                    const char* out_file, const char* dir = NULL);
```

```
   // загрузка файла с FTP-сервера
   int GetFTPFileEx ( const char* ftp_address, const char* in_file,
                      const char* out_file, const char* dir = NULL);
   // установить прогресс-бар для получения текущего состояния
   void SetProgress ( HWND hProgressBar, bool bSet);
   // установить текстовое окно для вывода текущего состояния
   void SetStatusView ( HWND hText, bool bSet);
   // проверка модемного соединения с сервером
   bool IsModemConnection () { return m_dwStateConnection &
                                  INTERNET_CONNECTION_MODEM; }
   // проверка соединения в локальной сети
   bool IsLANConnection () { return m_dwStateConnection &
                                INTERNET_CONNECTION_LAN; }
   // проверка правильности соединения в локальной сети
   bool IsLANValid () { return m_dwStateConnection &
                           INTERNET_CONNECTION_CONFIGURED; }
   // проверка наличия прокси-сервера
   bool IsProxy () { return m_dwStateConnection &
                       INTERNET_CONNECTION_PROXY; }
   // проверка поддержки службы удаленного соединения
   bool IsRAS () { return m_dwStateConnection &
                     INTERNET_RAS_INSTALLED; }
   // проверка автономного режима
   bool IsOffline () { return m_dwStateConnection &
                         INTERNET_CONNECTION_OFFLINE; }
private:
   HINTERNET m_hInternet; // дескриптор Интернета
   DWORD m_dwStateConnection; // состояние соединения с сетью
   HWND m_hProgress; // дескриптор прогресс-бара
   HWND m_hText; // дескриптор окна для вывода текста
   bool m_bIsProgress; // включение контроля загрузки файла
   bool m_bIsStatus; // включение информации о загрузке файла
}; // окончание класса
```

Листинг 21.11. Файл DownloadFile.cpp класса `DownloadFile`

```
#include "stdafx.h"
#include "DownloadFile.h"
#include <stdio.h>
#include <commctrl.h>
// реализация класса
DownloadFile :: DownloadFile ()
```

```
(
   // инициализируем переменные класса
   m_hInternet = NULL;
   m_dwStateConnection = 0;
   m_hProgress = NULL;
   m_bIsProgress = false;
   m_bIsStatus = false;
   m_hText = NULL;
}
DownloadFile :: ~DownloadFile ()
{
   // закрываем дескриптор Интернета
   if ( m_hInternet) InternetCloseHandle ( m_hInternet);
   m_hInternet = NULL;
}
bool DownloadFile :: Initialize ( bool bConnect)
{
   // проверяем состояние соединения с сетью
   if ( !InternetGetConnectedState ( &m_dwStateConnection, 0))
   {
     // если надо, выводим на экран окно установки соединения
     if ( bConnect)
     {
       if ( InternetAttemptConnect ( 0) != ERROR_SUCCESS)
            return false;
     }
     else // иначе выходим с ошибкой
            return false;
   }
   // если автономный режим, выходим из функции
   if( !m_dwStateConnection & INTERNET_CONNECTION_OFFLINE)
        return false;
   // инициализируем функции Интернета для нашей программы
   m_hInternet = InternetOpen ( "MyDownloadFile",
                 INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
   if ( m_hInternet == NULL)
        return false; // ошибка инициализации
   return true; // ошибок нет
}
void DownloadFile :: SetStatusView ( HWND hText, bool bSet)
{
   if ( !m_hInternet) return;
   if ( !hText) return;
```

```
   m_hText = hText; // дескриптор окна для вывода информации
   m_bIsStatus = bSet; // включение информации о состоянии
}
void DownloadFile :: SetProgress ( HWND hProgressBar, bool bSet)
{
   if ( !m_hInternet) return;
   if ( !hProgressBar) return;
   m_hProgress = hProgressBar; // дескриптор прогресс-бара
   m_bIsProgress = bSet; // включаем использование прогресс бара
}
int DownloadFile :: GetFile ( const char* address, const char* out_file)
{
   if ( !m_hInternet) return 0;
   // объявляем необходимые переменные
   FILE* pOut = NULL; // дескриптор файла
   HINTERNET hOpenServer = NULL; // дескриптор сервера
   char* pData = NULL; // указатель на буфер данных
   char m_szStatus[100]; // буфер состояния
   DWORD dwFileSize = 0, dwReadBytes = 0, dwCurRead = 0;
   DWORD  dwTimer = 100, dwFirst = 0, dwNext = 0;
   DWORD dwBufferLen = sizeof ( dwFileSize);
   // открываем указанный сетевой ресурс HTTP
   hOpenServer = InternetOpenUrl ( m_hInternet, address, NULL, 0,
          INTERNET_FLAG_DONT_CACHE | INTERNET_FLAG_PRAGMA_NOCACHE, 0);
   if ( hOpenServer == NULL) return 0; // если ошибка, выходим
   // только для HTTP получаем размер исходного файла
   if ( !HttpQueryInfo ( hOpenServer, HTTP_QUERY_CONTENT_LENGTH |
   HTTP_QUERY_FLAG_NUMBER, ( LPVOID) &dwFileSize, &dwBufferLen, NULL))
   {
     InternetCloseHandle ( hOpenServer); // закрываем ресурс
     return 0;
   }
   // выделяем память под буфер
   pData = new char [dwFileSize + 1];
   if ( pData == NULL)
    {
     InternetCloseHandle ( hOpenServer); // закрываем ресурс
     return 0;
   }
   // открываем файл для записи
   if  ( ( pOut = fopen ( out_file, "wb")) == NULL)
   {
     delete [] pData; // освобождаем память
```

```
    pData = NULL;
    InternetCloseHandle ( hOpenServer); // закрываем ресурс
    return 0;
  }
  // если включен вывод информации о состоянии
  if ( m_bIsStatus)
     // получаем текущее значение работы системы в миллисекундах
     dwFirst = GetTickCount ();
  // если включен прогресс-бар, инициализируем его
  if ( m_bIsProgress)
  SendMessage ( m_hProgress, PBM_SETRANGE, 0,
                MAKELPARAM ( 0, dwFileSize));
  // запускаем цикл для чтения файла из сети
  do
  {
    // читаем данные из сети
    if ( !InternetReadFile ( hOpenServer, pData, sizeof ( pData),
                             &dwReadBytes))
    {
      fclose ( pOut); // закрываем файл
      delete [] pData; // освобождаем память
      pData = NULL;
      InternetCloseHandle ( hOpenServer); // закрываем ресурс
      return 0;
    }
    // записываем прочитанные данные в файл на локальном диске
    if ( !dwFileSize) // если файл прочитан полностью, выходим
       break;
    else
       fwrite ( pData, sizeof ( char), dwReadBytes, pOut);
    // сбрасываем файловый кэш на диск
    fflush ( pOut);
    // вычисляем текущее число прочитанных байтов
    dwCurRead += dwReadBytes;
    // если включен прогресс-бар, отображаем новое положение
    if ( m_bIsProgress)
    {
      SendMessage ( m_hProgress, PBM_SETPOS, ( WPARAM) dwCurRead, 0);
      UpdateWindow ( m_hProgress);
    }
    // если включен вывод информации о текущем состоянии
    if ( m_bIsStatus)
```

```
    {
      // вычисляем число прочитанных байтов и скорость передачи
      float ft = 0;
      ft = ( float) dwCurRead;
      ft /= ( ( float) dwTimer) / 1000.0f;
      ft /= 1024.0f;
      // форматируем данные о состоянии
      sprintf ( m_szStatus, "Read: %u KB Total: %d KB \
      Transfer: %1.1f KB/s",  dwCurRead / 1024, dwFileSize / 1024, ft);
      // выводим данные на экран
      SendMessage ( m_hText, WM_SETTEXT, 0,
                  ( LPARAM) ( LPTSTR) m_szStatus);
      // обновляем окно вывода
      UpdateWindow ( m_hText);
      InvalidateRect ( m_hText, NULL, false);
      // получаем следующий отсчет времени
      dwNext = GetTickCount ();
      dwTimer = dwNext - dwFirst; // вычисляем промежуток
      if ( dwTimer == 0) dwTimer = 100;
    }
    // уменьшаем остаток непрочитанных данных
    dwFileSize -= dwReadBytes;
  } while ( 1);
  pData[dwFileSize] = NULL;
  // восстанавливаем, если нужно, начальное состояние прогресс-бара
  if ( m_bIsProgress)
     SendMessage ( m_hProgress, PBM_SETPOS, 0, 0);
  // закрываем файл
  if ( pOut) fclose ( pOut);
  // освобождаем память
  if ( pData) delete [] pData;
  pData = NULL;
  // закрываем ресурс
  InternetCloseHandle ( hOpenServer);
  return 1; // выходим из функции
}
int DownloadFile :: GetFTPFile ( const char* ftp_address,
        const char* in_file, const char* out_file, const char* dir)
{
  if ( !m_hInternet) return 0;
  HINTERNET hOpenServer = NULL; // дескриптор сервера
  // открываем ресурс на сервере FTP
  hOpenServer = InternetConnect (  m_hInternet, ftp_address,
    INTERNET_DEFAULT_FTP_PORT, NULL, NULL, INTERNET_SERVICE_FTP, 0, 0);
```

```
   if ( !hOpenServer) return 0; // если ошибка, выходим
   if ( dir)
   {
     // если нужно, устанавливаем текущую папку на сервере FTP
     if (!FtpSetCurrentDirectory ( hOpenServer, dir))
     {
        InternetCloseHandle ( hOpenServer);
        return 0;
     }
   }
   // читаем файл с сервера FTP
   if ( !FtpGetFile ( hOpenServer, in_file, out_file, FALSE,
         FILE_ATTRIBUTE_NORMAL,
         FTP_TRANSFER_TYPE_BINARY | INTERNET_FLAG_RELOAD, 0))
   {
     InternetCloseHandle ( hOpenServer); // закрываем ресурс
     return 0;
   }
   InternetCloseHandle ( hOpenServer); // закрываем ресурс
   return 1; // выходим из функции
}
int DownloadFile :: GetFTPFileEx ( const char* ftp_address,
     const char* in_file, const char* out_file, const char* dir)
{
   if ( !m_hInternet) return 0;
   // объявляем переменные
   HINTERNET hOpenServer = NULL, hFtpFile = NULL;
   DWORD  dwTimer = 100, dwFirst = 0, dwNext = 0;
   DWORD dwCurRead = 0, dwReadBytes  = 0;
   char buffer[1024]; // буфер для данных
   char m_szStatus[100]; // буфер для вывода информации
   FILE* pOut = NULL; // дескриптор файла
   // открываем ресурс на сервере FTP
   hOpenServer = InternetConnect (  m_hInternet, ftp_address,
              INTERNET_DEFAULT_FTP_PORT, NULL, NULL, NTERNET_SERVICE_FTP,
              0, 0);
   if ( !hOpenServer) return 0; // если ошибка, выходим
   if ( dir)
   {
     // если нужно, устанавливаем текущую папку на сервере FTP
     if (!FtpSetCurrentDirectory ( hOpenServer, dir))
     {
        InternetCloseHandle ( hOpenServer);
```

```
      return 0;
    }
  }
  // открываем файл для чтения на сервере FTP
  if ( !( hFtpFile = FtpOpenFile ( hOpenServer, in_file, GENERIC_READ,
     FTP_TRANSFER_TYPE_BINARY | INTERNET_FLAG_TRANSFER_BINARY, 0)))
  {
    InternetCloseHandle ( hOpenServer);
     return 0;
  }
  // открываем файл для записи на локальном диске
  if  ( ( pOut = fopen ( out_file, "wb")) == NULL)
  {
    // освобождаем ресурсы
    InternetCloseHandle ( hFtpFile);
    InternetCloseHandle ( hOpenServer);
    return 0;
  }
  // если включен вывод информации о состоянии
  if ( m_bIsStatus)
     // получаем текущее значение работы системы в миллисекундах
     dwFirst = GetTickCount ();
  // если у вас установлена VC++.NET или обновлены файлы,
  // можно получить размер исходного файла, как показано ниже
  // DWORD dwSizeHi = 0, dwSizeLo = 0, dwFileSize = 0;
  // dwSizeLo = FtpGetFileSize ( hFtpFile, &dwSizeHi);
  // dwFileSize = dwSizeLo | ( dwSizeHi << 32);
  // далее следует изменить код как в функции GetFile
  // если у вас VC++ 6.0, определение размера файла опускаем
  // запускаем цикл для чтения файла из сети
  do
  {
    // читаем данные из файла на сервере FTP
    if ( !InternetReadFile ( hFtpFile, buffer, 1024, &dwReadBytes))
         break;
    // записываем данные в файл на локальном диске
    if ( !dwReadBytes)
          break;
    else
          fwrite ( buffer, sizeof ( char), dwReadBytes, pOut);
    // сбрасываем файловый кэш на диск
    fflush ( pOut);
    // вычисляем текущее число прочитанных байтов
    dwCurRead += dwReadBytes;
```

```
    // если включен вывод информации о текущем состоянии
    if ( m_bIsStatus)
    {
      // вычисляем число прочитанных байтов и скорость пердачи
      float ft = 0;
      ft = ( float) dwCurRead;
      ft /= ( ( float) dwTimer) / 1000.0f;
      ft /= 1024.0f;
      // форматируем данные о состоянии
      sprintf ( m_szStatus, "Read: %u KB Transfer: %1.1f KB/s",
                dwCurRead / 1024, ft);
      // выводим данные на экран
      SendMessage ( m_hText, WM_SETTEXT, 0,
                  ( LPARAM) ( LPTSTR) m_szStatus);
      // обновляем окно вывода
      UpdateWindow ( m_hText);
      InvalidateRect ( m_hText, NULL, false);
      // получаем следующий отсчет времени
      dwNext = GetTickCount ();
      dwTimer = dwNext - dwFirst; // вычисляем промежуток
      if ( dwTimer == 0) dwTimer = 100;
    }
  } while ( 1);
  // закрываем файл
  if ( pOut) fclose ( pOut);
  // закрываем ресурсы
  InternetCloseHandle ( hFtpFile);
  InternetCloseHandle ( hOpenServer);
  return 1; // выходим из функции
}
```

Класс `DownloadFile` достаточно компактен и прост, но позволяет вполне свободно скачивать из Интернета практически любые файлы: html, zip, rar, exe и т. п.

Перед использованием класса всегда следует вызывать функцию `Initialize`, которая проверяет состояние подключения к сети и инициализирует ресурсы доступа к сетевым ресурсам. После этого можно с помощью функций `SetProgress` и `SetStatusView` включить возможность проверки в реальном времени процесса загрузки файла. Для удобства дополнительно отслеживается текущая скорость выполнения операции.

Чтобы скопировать файл из сети, нужно выбрать соответствующую функцию: `GetFile` (копирование файлов с сервера HTTP), `GetFTPFile` (копирова-

ние файлов с сервера FTP без вывода информации о ходе процесса) или GetFTPFileEx (то же самое, но с выводом текущей информации о состоянии).

Для загрузки файлов по протоколу HTTP выполняем следующие шаги:

1. Открываем сетевой ресурс (HTTP или FTP) посредством библиотечной функции InternetOpenUrl.
2. Определяем размер исходного файла с помощью функции HttpQueryInfo. Данная функция не поддерживает работу с серверами FTP. Для них существует новая (для Internet Explorer версии 5.0 и выше) библиотечная функция — FtpGetFileSize. В VC++ .NET она включена в файл WinInet.h. Программисты, работающие в VC++ 6.0, могут обновить указанный файл или в общих настройках среды разработчика установить новый путь к подключаемым файлам.
3. Открываем файл для записи на локальный диск.
4. Поблочно (размер задается буфером) считываем сетевой файл и сохраняем на локальном диске. Для чтения файла применяется функция InternetReadFile.
5. После того как весь исходный файл прочитан, закрываем целевой файл на диске и освобождаем открытый сетевой ресурс функцией InternetCloseHandle.

Для получения файла, размещенного на файловом сервере FTP, достаточно выполнить несколько операций:

1. Открываем новый сеанс связи с сервером (FTP или HTTP), используя функцию InternetConnect.
2. Если искомый файл расположен в каталоге, предварительно следует открыть этот каталог посредством функции FtpSetCurrentDirectory.
3. С помощью функции FtpGetFile считываем файл с сервера. Данная функция не вернет управление программе, пока весь файл не будет прочитан или пока не произойдет ошибка.
4. Освобождаем открытый сетевой ресурс функцией InternetCloseHandle.

И напоследок приведу примеры работы с классом DownloadFile (листинг 21.12).

Листинг 21.12. Примеры использования класса DownloadFile

```
#include "DownloadFile.h"
// получаем дескрипторы элементов для слежения за ходом процесса
HWND hProgressBar = GetDlgItem ( hDlg, IDC_MYPROGRESS);
HWND hStatusText = GetDlgItem ( hDlg, IDC_MYSTATUSTEXT);
```

```
// объявляем класс
DownloadFile down;
// инициализируем функции Интернета и при необходимости вызываем
// стандартное диалоговое окно для установки соединения
down.Initialize ( true);
// включаем использование прогресс-бара
down.SetProgress ( hProgressBar, true);
// включаем использование статистики
down.SetStatusView ( hStatusText, true);
// загружаем файл с сервера и сохраняем на локальный диск с новым именем
down.GetFile ( "http://www.mysite.com/book.zip", "C:\\kniga.zip");
// отключаем использование прогресс-бара
down.SetProgress ( hProgressBar, false);
// загружаем музыкальный файл в формате MP3 с файлового сервера
// из каталога "pub/mp3files"
down.GetFTPFileEx ( "ftp.musik.com", "madonna.mp3", "C:\\madonna.mp3",
                    "pub/mp3files");
```

Вообще, набор функций для программирования Интернета гораздо шире представленного здесь. Думаю, читателям будет интересно самостоятельно изучить их, тем более что простота работы с ними доставляет поистине незабываемое удовольствие.

ГЛАВА 22

Почта и сеть

Всем, кто использует операционную систему Windows, известна "небольшая" программа Outlook Express, позволяющая получать и отправлять почтовые сообщения, проще говоря, письма. Одним она нравится, другим нет. Судить о ее достоинствах или недостатках я не берусь, каждый делает свой выбор сам, но что делать, если программист хочет написать собственную почтовую программу или просто добавить в уже созданный программный продукт поддержку работы с почтой? Если во втором случае можно просто вызвать стандартную почтовую программу (тот же Outlook Express) с помощью функции `ShellExecute` *(см. гл. 19)*, то в первом придется искать другие возможности. Вот об этом и пойдет речь в данной главе. Кроме этого, мы рассмотрим важные моменты программирования сетей, неразрывно связанные с почтовыми службами.

Для удобства восприятия информации все основные темы можно представить в виде списка отдельных тем:

1. Отправление почтового сообщения.
2. Получение почтового сообщения.
3. Использование сетевых функций

22.1. Отправление почтового сообщения

При работе с почтовыми сообщениями используется стандартный транспортный протокол SMTP (Simple Mail Transfer Protocol). Он традиционно базируется на управляющем протоколе передачи TCP (Transmission Control Protocol), где ему отведен порт (канал передачи) под номером 25. TCP поддерживает передачу 8-разрядных байтов. Данные же в протоколе SMTP построены на 7-разрядных символах ASCII, поэтому при передаче по каналу TCP в каждом байте старший бит устанавливается в 0. Данные передаются

с одного компьютера на другой непосредственно с использованием одинаковой транспортной службы, а также через один или более так называемых SMTP-серверов, подключенных с применением одинаковой транспортной службы. Для правильной работы промежуточного SMTP-сервера он должен иметь имя, совпадающее с именем почтового ящика адресата. Например, для почтового сервера Yandex назначено имя **yandex.ru** (полное имя **smtp.yandex.ru**), которое совпадает с любым почтовым ящиком, зарегистрированным на данном сервере (например, **noname@yandex.ru**). Работа по протоколу SMTP происходит с помощью специальных команд. Синтаксис этих команд четко оговорен стандартом. После команды могут следовать дополнительные параметры, а в завершении обязательно должны размещаться два управляющих символа: возврат каретки (код 0D) и новая строка (код 0A). В конце основного текста письма необходимо разместить две пары управляющих символов с точкой посередине.

Кроме управляющих команд, протокол поддерживает специальные числовые коды ответов, позволяющие обнаружить ошибки и сбои при передаче почтовых сообщений. Основные команды протокола SMTP перечислены в табл. 22.1.

В приведенной таблице указаны полные названия команд, а в скобках приводятся их сокращенные версии, непосредственно используемые в работе. Числовые коды ответов и их описание представлены в табл. 22.2.

Таблица 22.1. Список основных команд протокола SMTP

Имя команды (принятое сокращение)	Описание
HELLO (HELO)	Позволяет указать имя хоста (адресуемого SMTP-сервера)
EXT HELLO (EHLO)	Позволяет использовать дополнительные опции для управления почтовым сервером
MAIL (MAIL)	Позволяет указать адрес одного или нескольких отправителей почтового сообщения
RECIPIENT (RCPT)	Позволяет указать адрес получателя почтового сообщения
DATA (DATA)	Позволяет поместить передаваемые данные почтового сообщения в буфер
SEND (SEND)	Служит для инициализации передачи почтового сообщения одному или нескольким терминалам
SEND OR MAIL (SOML)	Служит для инициализации передачи почтового сообщения одному или нескольким терминалам или почтовым ящикам

Таблица 22.1 (окончание)

Имя команды (принятое сокращение)	Описание
SEND AND MAIL (SAML)	Служит для инициализации передачи почтового сообщения одному или нескольким терминалам и почтовым ящикам
RESET (RSET)	Позволяет прервать текущую передачу почтового сообщения
VERIFY (VRFY)	Позволяет проверить правильность идентификации адресата
EXPAND (EXPN)	Позволяет проверить правильность идентификации списка адресатов
HELP (HELP)	Позволяет обрабатывать различную справочную информацию
NOOP (NOOP)	Пустая команда
QUIT (QUIT)	Позволяет установить для адресата передачу положительного ответа и закрыть канал связи
TURN (TURN)	Позволяет переопределить роль адресата на роль отправителя

Таблица 22.2. *Числовые коды ответов*

Код	Описание
211	System status, or system help reply (состояние системы)
214	Help message (справочное сообщение)
220	<domain> Service ready (служба доступна)
221	<domain> Service closing transmission channel (канал передачи закрыт службой)
250	Requested mail action okay, completed (требуемая почтовая операция успешно выполнена)
251	User not local; will forward to <forward-path> (для нелокального пользователя требуется указать полный адрес получателя)
354	Start mail input; end with <CRLF>.<CRLF> (можно начинать передачу почтового отправления)
421	<domain> Service not available, closing transmission channel (служба занята, закрытие канала передачи)
450	Requested mail action not taken: mailbox unavailable (операция не выполнена, почтовый ящик недоступен)

Таблица 22.2 (окончание)

Код	Описание
451	Requested action aborted: error in processing (ошибка обработки данных)
452	Requested action not taken: insufficient system storage (недостаточно памяти)
500	Syntax error, command unrecognized (неправильная команда)
501	Syntax error in parameters or arguments (недопустимые аргументы для команды)
502	Command not implemented (команда не поддерживается)
503	Bad sequence of commands (ошибка последовательности команд)
504	Command parameter not implemented (параметры команды не поддерживаются)
550	Requested action not taken: mailbox unavailable (почтовый ящик недоступен)
551	User not local; please try <forward-path> (для нелокального пользователя требуется указать полный адрес получателя)
552	Requested mail action aborted: exceeded storage allocation (ошибка памяти)
553	Requested action not taken: mailbox name not allowed (недопустимое имя почтового ящика)
554	Transaction failed (ошибка в работе)

Использование возвращаемых кодов помогает синхронизировать запросы и команды во время передачи почтового сообщения. Каждый код состоит из трех числовых символов, за которыми следует определенное текстовое сообщение.

В принципе, вот и все общие сведения о протоколе SMTP, с которыми мне хотелось познакомить читателя. Для программирования в Windows этих данных более чем достаточно. Если же вы используете, например, UNIX, следует более детально ознакомиться с документацией на протокол. В завершении приведу некоторые ограничения, оговоренные стандартом:

- максимальная длина имени домена не может превышать 64 символа;
- максимальная длина имени пользователя не может превышать 64 символа;
- максимальная длина обратного (адрес отправителя) или прямого (адрес получателя) пути не может превышать 256 символов, включая разделители и знаки пунктуации;
- максимальная длина командной строки вместе с управляющим кодом (2 байта) не может превышать 512 символов;

- максимальная длина получаемой в ответ строки с учетом ответных кодов и управляющего кода не может превышать 512 символов;
- максимальная длина текстового сообщения с учетом управляющего кода не может превышать 1 000 символов;
- максимальное количество получателей, использующих буферизацию, ограничено числом 100.

Для организации передачи почтового сообщения можно воспользоваться набором стандартных функций Win32 API для работы с сокетами. Чтобы подключить к проекту эти функции, достаточно добавить в начало кода файл определений Winsock.h, а в опциях компоновщика определить ссылку на библиотеку Ws2_32.lib. Мы не будем создавать целый класс, вполне хватит одной функции, которая и возьмет на себя все управление пересылкой почтового сообщения на удаленный сервер сети. Пример такой функции представлен в листинге 22.1.

Листинг 22.1. Отправка почтового сообщения по протоколу SMTP

```
#include "stdafx.h"
#include <winsock.h>
#include <stdio.h> // функция sprintf
// объявляем функции
bool SendMail (); // пересылка письма
bool InitSocket ( BYTE major, BYTE minor); // инициализация библиотеки
// функция для отправки письма
bool SendMail ()
{
   // готовим структуру письма
   const char* msg[] = "Date: 1 April 2005 10:11:12\r\n \
       From: noname@server.ru\r\nTo: mybox@yandex.ru\r\n \
       Subject: Privet\r\nПервое апреля - никому не верю.\r\n";
   // объявляем переменные
   SOCKET sock_smtp = NULL; // дескриптор сокета
   struct sockaddr_in address; // структура для хранения адреса сервера
   char status[300]; // буфер для возвращаемой сервером информации
   int iResult = 0; // результат операции
   // инициализируем библиотеку сокетов и проверяем ее версию
   if ( !InitSocket ( 1, 1)) return false;
   // создаем новый канал связи TCP
   sock_smtp = socket ( AF_INET, SOCK_STREAM, IPPROTO_TCP);
   // если канал не создан, выходим из функции
   if(sock_smtp == INVALID_SOCKET)
   {
     WSACleanup (); // выгружаем библиотеку сокетов Ws2_32.dll
```

```
  return false;
}
// заполняем адресную структуру
address.sin_port = htons ( 25); // номер порта
// сетевой адрес почтового сервера ( требуется получить у провайдера)
address.sin_addr.s_addr = inet_addr ( "123.233.232.100");
// тип адреса
address.sin_family = AF_INET;
// устанавливаем связь с сервером по созданному каналу
if ( connect ( sock_smtp, ( struct sockaddr*) &address,
               sizeof ( sockaddr_in)))
{
  // если ошибка
  closesocket ( sock_smtp); // закрываем канал связи
  WSACleanup (); // выгружаем библиотеку сокетов Ws2_32.dll
  return false;
}
// передаем письмо по установленному каналу связи
iResult = recv ( sock_smtp, status, 300, 0); // проверяем канал
if ( iResult == SOCKET_ERROR) goto error_end; // ошибка
status[iResult] = '\0'; // выделяем результат
OutputDebugString ( status); // выводим на консоль отладчика
// передаем имя хоста
sprintf ( status, "HELO %s\r\n", "yandex.ru");
send ( sock_smtp, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_smtp, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end;
status[iResult] = '\0';
OutputDebugString ( status);
// передаем почтовый адрес отправителя
sprintf ( status, "MAIL FROM:<%s>\r\n", "noname@server.ru");
send ( sock_smtp, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_smtp, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end;
status[iResult] = '\0';
OutputDebugString ( status);
// передаем почтовый адрес получателя
sprintf ( status, "RCPT TO:<%s>\r\n", mybox@yandex.ru);
send ( sock_smtp, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_smtp, status, 300, 0);
```

```
    if ( iResult == SOCKET_ERROR) goto error_end;
    status[iResult] = '\0';
    OutputDebugString ( status);
    // подготовка текста письма к отправке
    sprintf ( status, "DATA \r\n");
    send ( sock_smtp, status, strlen ( status), 0);
    // проверяем результат операции
    iResult = recv ( sock_smtp, status, 300, 0);
    if ( iResult == SOCKET_ERROR) goto error_end;
    status[iResult] = '\0';
    OutputDebugString ( status);
    // передаем текст письма
    sprintf ( status, "%s\r\n.\r\n", msg);
    send ( sock_smtp, status, strlen ( status), 0);
    // проверяем результат операции
    iResult = recv ( sock_smtp, status, 300, 0);
    if ( iResult == SOCKET_ERROR) goto error_end;
    status[iResult] = '\0';
    OutputDebugString ( status);
    // завершаем передачу
    sprintf ( status, "QUIT\r\n");
    send ( sock_smtp, status, strlen ( status), 0);
    // проверяем результат операции
    iResult = recv ( sock_smtp, status, 300, 0);
    if ( iResult == SOCKET_ERROR) goto error_end;
    status[iResult] = '\0';
    OutputDebugString ( status);
    // письмо успешно отправлено
    closesocket ( sock_smtp); // закрываем канал связи
    WSACleanup (); // выгружаем библиотеку сокетов Ws2_32.dll
    return true; // выходим из функции
// обработка ошибок
error_end:
    closesocket ( sock_smtp);
    WSACleanup ();
    return false; // выходим с ошибкой
}
// функция инициализации библиотеки сокетов
bool InitSocket ( BYTE major, BYTE minor)
{
    // добавляем информационную структуру для сокетов
    WSADATA socketInfo;
    // добавляем переменные для хранения кода ошибки и версии
    WORD wVersionSocket;
    int iError;
```

```
    // преобразуем составляющие требуемой версии
    wVersionSocket = MAKEWORD ( major, minor);
    // вызываем функцию WSAStartup
    iError = WSAStartup ( wVersionSocket, &socketInfo);
    if ( iError != 0)
     {
      // версия установленной библиотеки не подходит
      WSACleanup ();
      return false;
    }
    // версия библиотеки подходит, продолжаем работу
    return true;
}
```

Итак, прежде всего необходимо убедиться в наличии библиотеки сокетов и инициализировать ее для работы. Для этого вызываем функцию `WSAStartup`, передавая в качестве параметров старшее и младшее значения версии библиотеки (в данном случае 1.1). В случае если файл библиотеки (Ws2_32.dll) не удовлетворяет условию, выгружаем ресурсы сокетов с помощью функции `WSACleanup`.

После успешной инициализации создаем новый канал связи, используя функцию `socket`. В качестве параметров задаем тип протокола связи (TCP) и способ обмена данными. При условии успешного выполнения функция вернет дескриптор для вновь созданного сокета (канала связи). Далее выполняем настройку адреса и номера канала почтового сервера, заполняя поля структуры `sockaddr_in`. Номер канала всегда равен 25, а адрес сервера следует получить у провайдера услуг. Если все готово, устанавливаем соединение с помощью функции `connect` и передаем почтовое сообщение на удаленный сервер. Для этого вызываем функции `recv` и `send`. Первая функция служит для получения ответных сообщений, а вторая передает данные на почтовый сервер. Когда все данные переданы, закрываем канал связи (`closesocket`) и выгружаем библиотеку сокетов (`WSACleanup`). Вот, в принципе, и все сложности.

22.2. Получение почтового сообщения

Для получения почты и управления почтовым ящиком используется протокол POP3 (Post Office Protocol). Он позволяет динамически обрабатывать почтовые сообщения, расположенные на сервере. Как правило, для связи применяется канал под номером 110, поскольку POP3 так же, как и SMTP, базируется на транспортном протоколе TCP. Для управления почтой имеется собственный набор команд. Каждая команда состоит из ключевого слова, параметров и обязательных завершающих символов: возврат каретки (код `0D`)

и новая строка (код 0A). Ключевое слово всегда отделяется от параметров одним пробелом. Длина любого из параметров не должна превышать 40 символов. Протокол поддерживает два кода состояния, которые возвращаются сервером как результат операции: "+OK" (успешное выполнение) и "-ERR" (произошла ошибка). За кодом следует текстовое описание и символы (0D и 0A). Иногда текст содержит несколько строк. При этом каждая строка завершается символами возврата каретки и новой строки, а последняя — дополнительным символом точки (код 2E) и символами 0D и 0A. Список команд протокола POP3 представлен в табл. 22.3.

Таблица 22.3. *Список команд протокола POP3*

Имя команды	Описание
`DELE [N]`	Удаляет сообщение под номером N из почтового ящика
`LIST [N]`	Получение списка сообщений (с указанием порядкового номера и размера) или указанного сообщения с номером N
`NOOP`	Пустая команда
`PASS`	Ввод пароля пользователя
`QUIT`	Завершение связи
`RETR [N]`	Позволяет получить (повторно) сообщение с номером N
`RSET`	Восстанавливает все сообщения, помеченные для удаления
`STAT`	Позволяет получить текущее состояние почтового ящика (количество писем и используемый под сообщения размер ящика)
`TOP [N]`	Позволяет получить заголовок сообщения N и определить число строк в основном тексте
`UIDL [N]`	Возвращает уникальный строковый идентификатор для сообщения N
`USER`	Ввод имени пользователя

Дополнительные сведения можно получить в документации на протокол POP3.

Рассмотрим простой пример, позволяющий получить информацию о количестве почтовых сообщений и о используемом размере почтового ящика (листинг 22.2).

Листинг 22.2. Получение информации о состоянии почтового ящика

```
#include "stdafx.h"
#include <winsock.h>
#include <stdio.h> // функция sprintf
```

```
// объявляем функцию
bool CheckMailBox (); // проверка почтового ящика
// реализация
bool CheckMailBox ()
{
   SOCKET sock_pop3 = NULL; // дескриптор сокета
   struct sockaddr_in address; // структура для хранения адреса сервера
   hostent* info_server = NULL; // информация о сервере
   char status[300]; // буфер для обработки информации
   int iResult = 0;
   unsigned long pop_server = 0L;
   // обнуляем структуру sockaddr_in
   memset ( &address, 0, sizeof ( sockaddr_in));
   // инициализируем библиотеку сокетов и проверяем ее версию
   if ( !InitSocket ( 1, 1)) return false;
   // создаем новый канал связи
   sock_pop3 = socket ( AF_INET, SOCK_STREAM, IPPROTO_IP);
   // если канал не создан, выходим из функции
   if(sock_pop3 == INVALID_SOCKET)
   {
     WSACleanup (); // выгружаем библиотеку сокетов Ws2_32.dll
     return false;
   }
   // получаем адрес почтового сервера
   pop_server = inet_addr ( "pop.yandex.ru");
   // если ошибка
   if ( pop_server == 0xFFFFFFFF)
   {
     // пытаемся получить информацию о сервере
     info_server = gethostbyname ( "pop.yandex.ru");
     if ( !info_server) // нет данных
     {
       WSACleanup (); // выгружаем библиотеку сокетов Ws2_32.dll
       return false;
     }
     // сохраняем полученные данные
     memcpy ( &pop_server, info_server->h_addr_list[0], sizeof ( int));
   }
   // заполняем адресную структуру
   address.sin_family = AF_INET;
   address.sin_port = htons ( 110);  // номер канала для POP3
   memcpy ( &address.sin_addr, &pop_server, sizeof ( int));
```

```
// устанавливаем связь с сервером по созданному каналу
if ( connect ( sock_pop3, ( struct sockaddr*) &address,
               sizeof ( sockaddr_in)))
{
  closesocket ( sock_pop3); // закрываем канал связи
  WSACleanup (); // выгружаем библиотеку сокетов Ws2_32.dll
  return false;
}
// проверяем готовность сервера
iResult = recv ( sock_pop3, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end; // ошибка
status[iResult] = '\0'; // выделяем результат
OutputDebugString ( status); // выводим на консоль отладчика
// посылаем имя пользователя
sprintf ( status, "USER %s\r\n", "ivanov");
send ( sock_pop3, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_pop3, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end;
status[iResult] = '\0';
OutputDebugString ( status);
// посылаем пароль
sprintf ( status, "PASS %s\r\n", "seU78wda");
send ( sock_pop3, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_pop3, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end;
status[iResult] = '\0';
OutputDebugString ( status);
// получаем число почтовых сообщений и используемый размер ящика
sprintf ( status,"STAT %s\r\n");
send ( sock_pop3, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_pop3, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end;
status[iResult] = '\0';
OutputDebugString ( status);
// завершаем связь с сервером
sprintf ( status, "QUIT %s\r\n");
send ( sock_pop3, status, strlen ( status), 0);
// проверяем результат операции
iResult = recv ( sock_pop3, status, 300, 0);
if ( iResult == SOCKET_ERROR) goto error_end;
```

```
   status[iResult] = '\0';
   OutputDebugString ( status);
   // закрываем канал связи и выгружаем Ws2_32.dll
   closesocket ( sock_pop3);
   WSACleanup ();
   return true; // операция успешно выполнена
// обработка ошибок
error_end:
   closesocket ( sock_pop3);
   WSACleanup ();
   return false; // выходим с ошибкой
}
```

В представленном выше примере мы сначала устанавливаем соединение с почтовым сервером по протоколу POP3. После получения ответа посылаем имя пользователя (логин) и пароль. Далее с помощью команды `STAT` определяем количество почтовых сообщений и использованный размер (в байтах) ящика. В завершении передаем серверу команду `QUIT`. Вся информация, передаваемая почтовым сервером, выводится на консоль отладчика (функция `OutputDebugString`). Сделано это исключительно ради удобства восприятия и лучшего понимания работы протокола POP3. Думаю, читателю не составит труда самостоятельно выделить необходимые данные, а также проверить остальные команды POP3.

22.3. Использование сетевых функций

В завершение темы программирования сетей в операционных системах Windows поговорим о еще нескольких полезных возможностях. Прежде всего, я хочу познакомить читателей с тем, как можно определить IP-адрес сервера программным путем, зная только имя хоста (например, **smtp.yandex.ru**). Для этого можно применить функцию `WSAAsyncGetHostByName`. Она считывает указатель на структуру `hostent`, содержащую информацию о сервере: IP-адрес, официальное и альтернативные имена и тип возвращаемого адреса. Кроме того, функция возвращает дескриптор операции, с помощью которого можно отменить запрос (функция `WSACancelAsyncRequest`). Чтобы лучше понять, как все это работает, посмотрите листинг 22.3.

Листинг 22.3. Определение IP-адреса сервера

```
#include "stdafx.h"
#include <winsock.h>
// определяем собственное сообщение
#define MSG_SERVER_INFO    WM_USER + 7000
```

```
// определяем переменные
SOCKET sock_smtp = NULL; // дескриптор сокета
struct sockaddr_in address; // адресная структура
hostent* info_server = NULL; // информация о сервере
char* IP_server = NULL; // указатель на адрес сервера
char buffer[150]; // имя хоста
char szServerInfo[MAXGETHOSTSTRUCT];
// главная оконная функция ( в данном случае для диалога)
BOOL CALLBACK EmailDialog ( HWND hDlg, UINT msg, WPARAM wParam,
                             LPARAM lParam)
{
   switch ( msg) // обрабатываем сообщения
   {
    case WM_INITDIALOG: // инициализация диалога
    {
      // инициализируем библиотеку сокетов
      InitSocket ( 1, 1);
      // создаем сокет
      sock_smtp = socket ( AF_INET, SOCK_STREAM, IPPROTO_TCP);
      // считываем из окна редактирования имя хоста
      GetDlgItemText ( hDlg, IDC_MYEDITBOX, buffer, 150);
      // делаем асинхронный запрос серверу и регистрируем сообщение
      WSAAsyncGetHostByName ( hDlg, MSG_SERVER_INFO, buffer,
                              szServerInfo, MAXGETHOSTSTRUCT);
    }
    return true;
    case MSG_SERVER_INFO: // получаем сообщение от сервера
    (
       if ( HIWORD ( lParam) !=0) return 1; // ошибка
       // заполняем адресную структуру
       address.sin_family = AF_INET;
       address.sin_port = htons ( 25);  // номер канала для SMTP
       // сохраняем полученную информацию о сервере
       info_server = ( hostent*) &szServerInfo;
       // копируем адрес сервера
       memmove ( &IP_server, info_server->h_addr_list,
                 sizeof ( char*));
       // записываем адрес сервера в адресную структуру
       memmove ( &address.sin_addr.s_addr, &IP_server,
                 sizeof ( char*));
       // устанавливаем соединение с удаленным сервером
       connect ( sock_smtp, ( struct sockaddr*) &address,
                 sizeof ( sockaddr_in));
```

```
        // посылаем письмо на сервер
        // . . .
}
    break;
    case IDCANCEL: // закрываем диалоговое окно
      closesocket ( sock_smtp); // закрываем сокет
      WSACleanup (); // выгружаем библиотеку сокетов
      EndDialog ( hDlg, wParam);
      return true;
   }
   return false;
}
```

Как видно из примера, мы использовали функцию WSAAsyncGetHostByName для регистрации сообщения и получения информации о сервере. Поскольку функция работает в асинхронном режиме, она сразу возвращает управление программе. Когда запрашиваемая информация подготовлена сервером для передачи, программе посылается наше зарегистрированное ранее сообщение. При этом значение wParam принимает дескриптор выполняемой операции, а lParam содержит следующие значения: старшая 16-разрядная часть — код ошибки, младшая 16-разрядная часть — требуемый размер буфера для возвращаемых данных. После этого остается только обработать буфер szServerInfo и выделить адрес удаленного сервера. Далее применяем полученный адрес для установки связи с сервером и отправки почтового сообщения.

А теперь поговорим о том, как можно проверить занятость того или иного порта для протоколов TCP и UDP (User Datagram Protocol). Необходимость в этом может возникнуть не только в случае банального отправления почтового сообщения, но и для мониторинга сетевой активности, а также защиты компьютера от нежелательного доступа (или вирусов). Примеры реализации функций, проверяющих порты на локальном компьютере, представлены в листинге 22.4.

Листинг 22.4. Проверка занятости сетевого порта

```
#include "stdafx.h"
#include <winsock.h>
// определяем функции
int IsBusyTCPport ( unsigned short port);
int IsBusyUDPport ( unsigned short port);
// пишем реализацию
int IsBusyTCPport ( unsigned short port)
```

```
{
   // проверяем корректность номера порта
   if ( port > 65535) return -1;
   // инициализируем библиотеку сокетов
   if ( !InitSocket ( 1, 1)) return -2;
   // объявляем переменные
   SOCKET sock_port = NULL; // дескриптор сокета
   struct sockaddr_in address; // адресная структура
   // создаем новый сокет TCP
   sock_port = socket ( AF_INET, SOCK_STREAM, 0);
   // если ошибка, выходим из функции
   if ( sock_port == INVALID_SOCKET)
   {
     WSACleanup (); // выгружаем библиотеку сокетов
     return -3; // выходим из функции
   }
   // заполняем адресную структуру
   address.sin_family = AF_INET;
   address.sin_port = htons( port); // номер тестируемого порта
   // адрес локального сервера по умолчанию
   address.sin_addr.s_addr = inet_addr ( "127.0.0.1");
   // проверяем доступность указанного порта
   if ( connect ( sock_port, ( struct sockaddr*) &address,
                  sizeof ( sockaddr_in)))
   {
     closesocket ( sock_port);
     WSACleanup ();
     return 1; // порт занят или произошла ошибка
   }
   // закрываем сокет и выгружаем библиотеку из памяти
   closesocket ( sock_port);
   WSACleanup ();

   return 0; // порт не занят
}
int IsBusyUDPport ( unsigned short port)
{
   // проверяем корректность номера порта
   if ( port > 65535) return -1;
   // инициализируем библиотеку сокетов
   if ( !InitSocket ( 1, 1)) return -2;
   // объявляем переменные
   SOCKET sock_port = NULL; // дескриптор сокета
   struct sockaddr_in address; // адресная структура
```

```
  // создаем новый сокет UDP
  sock_port = socket ( AF_INET, SOCK_DGRAM, 0);
  // если ошибка, выходим из функции
  if ( sock_port == INVALID_SOCKET)
  {
    WSACleanup (); // выгружаем библиотеку сокетов
    return -3; // выходим из функции
  }
  // заполняем адресную структуру
  address.sin_family = AF_INET;
  address.sin_port = htons( port); // номер тестируемого порта
  // адрес локального сервера по умолчанию
  address.sin_addr.s_addr = inet_addr ( "127.0.0.1");
  // проверяем доступность указанного порта
  if ( bind ( sock_port, ( struct sockaddr*) &address,
                 sizeof ( sockaddr_in)))
  {
    closesocket ( sock_port);
    WSACleanup ();
    return 1; // порт занят или произошла ошибка
  }
  // закрываем сокет и выгружаем библиотеку из памяти
  closesocket ( sock_port);
  WSACleanup ();
  return 0; // порт не занят
}
```

Как вы, наверное, заметили, обе функции проверки порта очень похожи. Вначале мы проверяем, корректно ли задан номер тестируемого порта. Связано это с тем, что протоколы TCP и UDP поддерживают до 65 535 портов. Все номера портов распределены следующим образом: от 0 до 1 023 для стандартных служб, от 1 024 до 49 151 для заказных портов, 49 152—65 535 для частных и динамически распределяемых портов. Назначение некоторых часто используемых портов TCP показано в табл. 22.4.

Таблица 22.4. *Назначение стандартных портов*

Номер порта	Назначение
7	ECHO (тестирование сервера)
11	USERS (активные пользователи)
19	CHARGEN (генератор символов)

Таблица 22.4 (окончание)

Номер порта	Назначение
20	Протокол FTP (данные по умолчанию)
21	Протокол FTP (управление)
23	Протокол TELNET
25	Протокол SMTP
53	DOMAIN (служба доменных имен)
110	Протокол POP3
119	Протокол NNTP (Network News Transfer Protocol)
123	Протокол NTP (Network Time Protocol)
143	Протокол IMAP (Interim Mail Access Protocol)
161	Протокол SNMP (Simple Network Management Protocol)
563	Протокол NNTP с SSL
993	Протокол IMAP с SSL
995	Протокол POP3 с SSL

После проверки номера порта инициализируем библиотеку сокетов, вызывая функцию `InitSocket`. Далее создаем новый канал связи (`SOCK_STREAM` — TCP, `SOCK_DGRAM` — UDP) и пытаемся его открыть (назначить для UDP). Если функция `connect` (`bind` для UDP) возвращает нулевое значение, ошибок нет и указанный порт свободен, а поскольку нам требовалось лишь проверить состояние порта, закрываем его с помощью функции `closesocket`. Единственный момент, который мне хотелось бы уточнить, связан с указанием IP-адреса. В обоих случаях был задан стандартный локальный адрес 127.0.0.1, поскольку он является жестко фиксированным и предназначен для тестирования различных сетевых служб без установки соединения с сетью. Кроме того, он позволяет отлаживать различные сетевые компоненты и программы, заменяя реальный адрес сетевого сервера.

И напоследок, расскажу о том, как послать сообщение через популярную во всем мире службу ICQ. Прежде всего, нам понадобится адрес сайта (на данный момент это **web.icq.com**) и порт TCP под номером 80. Уникальный номер (идентификатор) пользователя службы ICQ следует узнать у того, кому вы будете передавать сообщение. Пример функции, реализующий отправку письма через ICQ, показан в листинге 22.5.

Листинг 22.5. Передача сообщения через службу ICQ

```
#include "stdafx.h"
#include <winsock.h>
// определяем функцию
int IsBusyTCPport ( unsigned short port);
// пишем реализацию
bool Send_ICQ_Message ( char* id, char* name, char* e_mail,
                        char* subject, char* msg)
{
   // объявляем переменные
   SOCKET sock_icq = NULL; // дескриптор сокета
   struct sockaddr_in address; // адресная структура
   hostent* info_icq = NULL; // информация о сервере
   char* p = NULL; // указатель на буфер данных
   int buf_len = 0; // размер буфера данных
   // обнуляем структуру sockaddr_in
   memset ( &address, 0, sizeof ( sockaddr_in));
   // вычисляем размер буфера данных
   buf_len = strlen ( id) + strlen ( name) + strlen ( e_mail) +
             strlen ( subject) +  strlen ( msg) + 100;
   // выделяем память под буфер
   p = new char [buf_len];
   if ( p == NULL) return false;

   // инициализируем библиотеку сокетов
   if ( !InitSocket ( 1, 1))
   {
     if ( p) delete [] p; // освобождаем память
     return false;
   }
   // определяем IP-адрес сервера службы ICQ
   info_icq = gethostbyname ( "web.icq.com");
   // проверяем результат
   if ( !info_icq)
   {
     WSACleanup ();
     if ( p) delete [] p; // освобождаем память
     return false; // выходим из функции с ошибкой
   }
   // создаем новый канал связи
    sock_icq = socket ( AF_INET, SOCK_STREAM, IPPROTO_IP);
```

```
// если канал не создан, выходим из функции
if ( sock_icq == INVALID_SOCKET)
{
  if ( p) delete [] p; // освобождаем память
  WSACleanup ();
  return false;
}
// заполняем адресную структуру
address.sin_family = AF_INET;
address.sin_port = htons( 80); // номер порта
address.sin_addr = *( in_addr*) host->h_addr; // адрес сервера
// устанавливаем соединение с удаленным сервером
if ( connect ( sock_icq, ( struct sockaddr*) &address,
               sizeof ( sockaddr_in)))
{
  closesocket ( sock_icq); // закрываем канал связи
  WSACleanup ();
  if ( p) delete [] p; // освобождаем память
  return false; // выходим из функции с ошибкой
}
// формируем сообщение
strcpy ( p, "GET /scripts/WWPMsg.dll?"); // копируем имя скрипта
strcat ( p, "from=");
strcat ( p, name); // добавляем имя отправителя сообщения
strcat ( p, "&fromemail=");
strcat ( p, e_mail); // добавляем адрес почты отправителя сообщения
strcat ( p, "&subject=");
strcat ( p, subject); // добавляем название темы сообщения
strcat ( p, "&body=");
strcat ( p, msg); // добавляем текст сообщения
strcat( p, "&to=");
strcat ( p, id); // добавляем идентификатор получателя
strcat( p, " HTTP/1.0\r\n"); // добавляем завершающую строку
// отправляем сообщение на сервер
send ( sock_icq, p, buf_len, 0);
// проверяем результат операции
recv ( sock_icq, p, buf_len, 0);
// выводим на консоль отладчика
OutputDebugString ( status);
// закрываем канал связи
closesocket ( sock_icq);
WSACleanup (); // выгружаем библиотеку сокетов
if ( p) delete [] p; // освобождаем память
```

```
   return true; // выходим из функции
}
// пример использования функции Send_ICQ_Message
char* text = "Hello friend"; // текст сообщения
// вызываем функцию
Send_ICQ_Message ( "2534888", "Ivan", "ivan@hotmail.com", "Privet",
                   text);
```

Итак, чтобы послать сообщение, мы выполнили следующие действия:

1. Инициализировали библиотеку сокетов с помощью функции `InitSocket`.
2. Определили IP-адрес удаленного сервера через функцию `gethostbyname`.
3. Создали новый канал связи по протоколу TCP (функция `socket`).
4. Установили связь с удаленным сервером посредством `connect`.
5. Сформировали и передали сообщение пользователю службы ICQ.

Кроме того, для наглядности мы распечатали информацию, полученную от сервера в окне отладчика, применив для этого функцию `OutputDebugString`. И хотя функция `Send_ICQ_Message` получилась немного громоздкой, но зато она позволяет легко корректировать основные параметры сообщения.

ГЛАВА 23

Трюки и секреты

В этой главе я собрал разнообразные полезные мелочи, которые, надеюсь, будут очень кстати начинающим и опытным программистам, работающим в Windows. Поскольку рассматриваемые здесь темы узки и разноплановы, их не удалось озвучить в предыдущих главах, но их невероятная привлекательность все же не позволила полностью от них отказаться. Каждая из представленных ниже тем абсолютно независима, поэтому читатель может сразу переходить к наиболее интересному для него разделу.

23.1. Определение параметров оборудования

Здесь мы поговорим об одной хитрой возможности, позволяющей достаточно легко получить конфигурационные параметры установленного в системе оборудования. К таким параметрам относятся, в первую очередь, номера портов ввода-вывода, выделенные прерывания и адреса в памяти. Все, что нам требуется, — это написать несколько функций для доступа к системному реестру. Именно там, в удобной, но несколько запутанной форме расположены все основные сведения о компьютере. Увидеть эти данные можно как с помощью диспетчера оборудования, так и при непосредственном просмотре файла реестра.

Сразу замечу, что размещение данных в реестре (пути), описывающих имеющиеся устройства, отличается для профессиональных систем Windows NT/2000/XP/SR3. Кроме того, доступ к реестру в профессиональных системах организовать несколько сложнее, чем в пользовательских системах Windows 9*x*/ME.

Данные о конфигурации оборудования находятся в следующих разделах реестра: **HKEY_DYN_DATA\Config Manager\Enum** для Windows 9*x*/ME и **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Enum** для Win-

dows NT/2000/XP/SR3. В подразделе **Enum** перечислены все доступные системе устройства и их основные параметры. В первую очередь, нас интересуют расположение устройства (параметр **HardWareKey**) и выделенные системой ресурсы (параметр **Allocation**). С помощью значения **HardWareKey** мы определим имя и класс устройства, а двоичный массив **Allocation** поможет вычислить порты ввода-вывода, адреса памяти, каналы IRQ и DMA. Остальные параметры в данном случае не имеют значения.

Итак, напишем функцию, которая сканирует реестр в поисках информации об установленном оборудовании для Windows 9*x*/ME. Пример такой функции показан в листинге 23.1.

Листинг 23.1. Определение основных параметров устройств в Windows 9*x*/ME

```
#include "stdafx.h"
// определяем макросы
// раздел реестра HKEY_DYN_DATA
#define HKDD_ENUM   "Config Manager\\Enum"
#define HKLM_ENUM   "Enum\\" // раздел реестра HKEY_LOCAL_MACHINE
#define MAX_DEVICES 200 // возможное количество устройств в системе
// перечисляем основные типы данных
enum DataTypes9x
{
   Memory_Type_9x = 0x00000001, // адрес в памяти
   Port_Type_9x = 0x00000002, // адрес порта ввода-вывода
   DMA_Type_9x = 0x00000003, // номер канала DMA
   IRQ_Type_9x = 0x00000004 // номер прерывания
};
// определяем структуры для описания данных
typedef struct _Memory9x // обработка адресов
{
   DWORD cbSize; // размер данных
   DWORD dwType; // тип данных
   DWORD Reserved; // не используется
   long int Base_Memory; // базовый адрес
   long int End_Memory; // завершающий адрес
   DWORD Reserved2; // не используется
   long int Align_Memory; // выравнивание данных
   long int CountBytes; // количество выделенных байтов в памяти
   long int MinAddrres; // минимальное значение адреса
   long int MaxAddrres; // максимальное значение адреса
   DWORD Reserved3; // не используется
} MEMORY9X, *PMEMORY9X;
```

```
// структура для хранения портов ввода-вывода
typedef struct _Port9x
{
   DWORD cbSize; // размер данных
   DWORD dwType; // тип данных
   DWORD Reserved; // не используется
   WORD Base_Port; // адрес базового порта
   WORD End_Port; // адрес последнего порта
   DWORD Reserved2; // не используется
   WORD IOR_Align; // выравнивание данных
   WORD CountPorts; // количество портов
   WORD IOR_Min; // минимальный адрес порта
   WORD IOR_Max; // максимальный адрес порта
   DWORD Reserved3; // не используется
} PORT9X, *PPORT9X;
// структура для хранения канала DMA
typedef struct _DMA9X
{
   DWORD cbSize; // размер данных
   DWORD dwType; // тип данных
   BYTE Reserved; // не используется
   BYTE Channel; // номер канала DMA
   WORD Reserved2; // не используется
} DMA9X, *PDMA9X;
// структура для хранения номера прерывания
typedef struct _IRQ9X
{
   DWORD cbSize; // размер данных
   DWORD dwType; // тип данных
   WORD Reserved; // не используется
   WORD Number; // номер прерывания
   DWORD Reserved2; // не используется
} IRQ9X, *PIRQ9X;
// структура для хранения имени и класса устройства
typedef struct _DeviceParams9x
{
   char HardWareKey[MAX_PATH]; // путь к устройству в реестре
   char DeviceDesc[150]; // название устройства
   char Class[50]; // класса устройства
   DWORD Base_Memory[16]; // базовые адреса в памяти
   DWORD End_Memory[16]; // завершающие адреса в памяти
   WORD Base_Port[16]; // адреса базовых портов ввода-вывода
```

```
   WORD End_Port[16]; // номера завершающих портов ввода-вывода
   BYTE DMA_Channel[8]; // номера выделенных каналов DMA
   BYTE IRQ_Number[16]; // номера выделенных прерываний
} DEVICEPARAMS9X, *PDEVICEPARAMS9X;
// определяем функцию поиска устройств в реестре
void GetDevicesConfig_9X ();
// массив структур DEVICEPARAMS9X для описания устройств
DEVICEPARAMS9X devices[MAX_DEVICES];
// реализация функции
void GetDevicesConfig_9X ()
{
   HKEY phKey = NULL, phKey2 = NULL; // дескрипторы разделов реестра
   DWORD dwKey = 0; // счетчик разделов реестра
   DWORD dwSize = 0; // размер значения параметра реестра
   DWORD dwTypeParametr = REG_SZ; // тип параметра реестра
   char section_name[MAX_PATH]; // имя найденного раздела
   char new_path[MAX_PATH]; // полный путь к разделу
   char hard_key[150]; // описание параметра
   int i = 0; // счетчик циклов
   BYTE byBuffer[300]; // буфер для данных
   // счетчики для определенных типов данных
   unsigned int uMemory_index = 0, uPort_index = 0, uDMA_index = 0,
                uIRQ_index = 0;
   MEMORY9X temp9x; // временный буфер для форматированных данных
   // структуры для хранения определенных данных
   MEMORY9X memory9x;
   PORT9X port9x;
   DMA9X dma9x;
   IRQ9X irq9x;
   // обнуляем структуры
   ZeroMemory ( &memory9x, sizeof ( MEMORY9X));
   ZeroMemory ( &port9x, sizeof ( PORT9X));
   ZeroMemory ( &dma9x, sizeof ( DMA9X));
   ZeroMemory ( &irq9x, sizeof ( IRQ9X));
   // обнуляем главную структуру данных
   for ( i = 0; i < MAX_DEVICES; i++)
     ZeroMemory ( &devices[i], sizeof ( DEVICEPARAMS9X));
   // открываем раздел реестра "HKEY_DYN_DATA\Config Manager\Enum"
   if( !RegOpenKeyEx ( HKEY_DYN_DATA, HKDD_ENUM, 0, KEY_READ, &phKey)
                    == ERROR_SUCCESS)
     return; // выходим из функции в случае ошибки
   while ( 1) // обрабатываем все найденные подразделы
```

```
{
  // получаем очередное имя подраздела
  if ( RegEnumKey ( phKey, dwKey, section_name, MAX_PATH)
                    == ERROR_NO_MORE_ITEMS)
     break; // если все подразделы найдены, завершаем цикл
  // собираем полный путь к найденному подразделу
  strcpy ( new_path, HKDD_ENUM);
  strcat ( new_path, "\\");
  strcat ( new_path, section_name);
  // открываем подраздел
  if ( RegOpenKeyEx ( HKEY_DYN_DATA, new_path, 0, KEY_QUERY_VALUE,
                      &phKey2)== ERROR_SUCCESS)
  {
    // предварительно определяем размер возвращаемых данных
    if ( RegQueryValueEx ( phKey2, "HardWareKey", NULL,
                 &dwTypeParametr,  NULL, &dwSize) == ERROR_SUCCESS)
    {
      // получаем путь к устройству
      RegQueryValueEx ( phKey2, "HardWareKey", NULL, NULL,
                        ( LPBYTE) hard_key, &dwSize);
      // сохраняем полученное значение
      strcpy ( devices[dwKey].HardWareKey, hard_key);
    }
    // определяем размер следующего значения
    if ( RegQueryValueEx ( phKey2, "Allocation", NULL,
                 &dwTypeParametr,  NULL, &dwSize) == ERROR_SUCCESS)
    {
       // получаем параметры устройства
       RegQueryValueEx ( phKey2, "Allocation", NULL, NULL,
                         ( LPBYTE) byBuffer, &dwSize);
       // восстанавливаем и форматируем данные
       int index = 8;
       ZeroMemory ( &temp9x, sizeof ( MEMORY9X));
       // анализируем данные и выделяем нужное
       while ( byBuffer[index] > 0)
       {
         // помещаем данные в структуру
         memmove ( ( void*) &temp9x, &byBuffer[index], 8);
         // выделяем тип данных
         switch ( temp9x.dwType)
         {
          case Memory_Type_9x: // адреса в памяти
            if  ( uMemory_index < 16)
```

```
{
  // помещаем данные о памяти в структуру
  memmove ( ( void*) &memory9x, &byBuffer[index],
               sizeof ( MEMORY9X));
  // сохраняем базовый адрес
  devices[dwKey].Base_Memory[uMemory_index] =
                           memory9x.Base_Memory;
  // сохраняем завершающий адрес
  devices[dwKey].End_Memory[uMemory_index] =
                           memory9x.End_Memory;
  uMemory_index++; // увеличиваем счетчик
  // обнуляем структуру
  ZeroMemory ( &memory9x, sizeof ( MEMORY9X));
 }
 break;
case Port_Type_9x: // адреса портов ввода-вывода
 {
  memmove ( ( void*) &port9x, &byBuffer[index],
               sizeof ( PORT9X));
  devices[dwKey].Base_Port[uPort_index] =
                             port9x.Base_Port;
  devices[dwKey].End_Port[uPort_index] = port9x.End_Port;
  uPort_index++;
  ZeroMemory ( &port9x, sizeof ( PORT9X));
 }
 break;
case DMA_Type_9x: // каналы прямого доступа к памяти
 {
  memmove ( ( void*)&dma9x, &byBuffer[index],
               sizeof ( DMA9X));
  devices[dwKey].DMA_Channel[uDMA_index] = dma9x.Channel;
  uDMA_index++;
  ZeroMemory ( &dma9x, sizeof ( DMA9X));
 }
 break;
case IRQ_Type_9x: // номер прерывания
 {
  memmove ( ( void*) &irq9x, &byBuffer[index],
               sizeof ( IRQ9X));
  devices[dwKey].IRQ_Number[uIRQ_index] = irq9x.Number;
  uIRQ_index++;
  ZeroMemory ( &irq9x, sizeof ( IRQ9X));
 }
```

```
              break;
            }
          // увеличиваем смещение в буфере данных
          index = index + temp9x.cbSize;
          }
        }
        // закрываем подраздел
        if ( phKey2) RegCloseKey ( phKey2);
      }
      else
      {
        // если не удалось открыть подраздел, переходим к следующему
        continue;
      }
      // увеличиваем счетчик разделов
      dwKey++;
      // обнуляем счетчики
      uMemory_index = 0;
      uPort_index = 0;
      uDMA_index = 0;
      uIRQ_index = 0;
    }
    // закрываем корневой раздел
    if ( phKey) RegCloseKey ( phKey);
    phKey = NULL;
    i = 0;
    // получаем описание найденного устройства
    for ( i; i < dwKey; i++)
    {
      // собираем путь к устройству
      strcpy ( new_path, HKLM_ENUM);
      strcat ( new_path, devices[i].HardWareKey);
      // открываем подраздел
      if ( RegOpenKeyEx ( HKEY_LOCAL_MACHINE, new_path, 0,
                          KEY_QUERY_VALUE, &phKey) == ERROR_SUCCESS)
      {
        // определяем размер значения параметра
        if ( RegQueryValueEx ( phKey, "DeviceDesc", NULL, &dwTypeParametr,
                               NULL, &dwSize) == ERROR_SUCCESS)
        {
          // получаем значение параметра
          RegQueryValueEx ( phKey, "DeviceDesc", NULL, NULL,
                            ( LPBYTE) hard_key, &dwSize);
```

```
        // сохраняем в основной структуре
        strcpy ( devices[i].DeviceDesc, hard_key);
      }
      // закрываем подраздел
      if ( phKey) RegCloseKey ( phKey);
    }
  }
  // определяем имя класса для найденных устройств
  for ( i = 0; i < dwKey; i++)
  {
    // собираем путь к устройству
    strcpy ( new_path, HKLM_ENUM);
    strcat ( new_path, devices[i].HardWareKey);
    // открываем подраздел
    if (RegOpenKeyEx ( HKEY_LOCAL_MACHINE, new_path, 0, KEY_QUERY_VALUE,
                       &phKey) == ERROR_SUCCESS)
    {
        // определяем размер значения параметра
        if ( RegQueryValueEx ( phKey, "Class", NULL, &dwTypeParametr,
                               NULL, &dwSize) == ERROR_SUCCESS)
        {
          RegQueryValueEx ( phKey, "Class", NULL, NULL,
                            ( LPBYTE) hard_key, &dwSize);
          strcpy ( devices[i].Class, hard_key);
        }
        if ( phKey) RegCloseKey ( phKey);
    }
  }
}
```

Для обработки данных были созданы пять специальных структур, четыре из которых хранят определенный тип данных, а последняя связывает все полученные данные в общий блок описателя устройства. После этого с помощью функции `RegOpenKeyEx` открыли раздел реестра **HKEY_DYN_DATA \Config Manager\Enum**. Данный раздел является динамическим, т. е. создается заново при каждом запуске операционной системы, поэтому не стоит полагаться на случай и сохранять текущую конфигурацию оборудования в файл или куда-то еще. Правильнее будет перед обращением к оборудованию повторно вызывать функцию `GetDevicesConfig_9X`. После успешного открытия раздела с помощью функции `RegEnumKey` получаем все имена подразделов. В качестве счетчика эта функция использует переменную `dwKey`, возвращая при каждом обращении новое значение. Когда все имена подразделов переданы, функция завершается с кодом `ERROR_NO_MORE_ITEMS`. В от-

крытом подразделе получаем значение параметра **HardWareKey** и сохраняем его в структуру DEVICEPARAMS. Этот параметр содержит определенный путь в реестре, и его значение понадобится нам позже для определения класса и типа устройства. Далее с помощью функции RegQueryValueEx считываем значение двоичного параметра **Allocation** (для получения сразу нескольких значений рекомендуется использовать функцию RegQueryMultipleValues). Этот параметр позволяет получить основные характеристики устройства: адреса доступа в памяти и адреса ввода-вывода, номер прерывания и канал DMA. И в завершение, восстанавливаем имя класса и тип для каждого найденного устройства.

А теперь рассмотрим, как аналогичную задачу можно решить в профессиональных системах Windows NT/2000/XP/SR3. Здесь, как я уже говорил, код будет несколько сложнее и запутаннее. Пример функции, определяющей доступные устройства, показан в листинге 23.2.

Листинг 23.2. Определение основных параметров устройств в Windows NT/2000/XP/SR3

```
#include "stdafx.h"
// путь к разделу в реестре, описывающему установленное оборудование
#define HKLM_ENUM_NT  "SYSTEM\\CurrentControlSet\\Enum"
#define MAX_DEVICES 200 // возможное количество устройств в системе
// дополнительные флаги
#define PORT_MEMORY 0x0000 // адрес порта в памяти
#define PORT_IO 0x0001 // адрес порта ввода-вывода
#define MEMORY_READ_WRITE 0x0000 // память для записи и чтения
#define MEMORY_READ_ONLY 0x0001 // память только для чтения
#define MEMORY_WRITE_ONLY 0x0002 // память только для записи
// тип канала DMA
#define DMA_8 0x0000
#define DMA_16 0x0001
#define DMA_32 0x0002
// типы данных
enum DataTypesNT
{
   Port_Type = 1, // адрес порта ввода-вывода
   IRQ_Type = 2, // номер прерывания
   Memory_Type = 3, // адрес в памяти
   DMA_Type = 4, // номер канала DMA
   BusNumber_Type = 6 // номер шины
};
// определяем тип значения для хранения адресов
typedef LARGE_INTEGER PHYSICAL_ADDRESS;
```

```
// определяем структуры для описания данных
#pragma pack ( 4)
typedef struct _IDDATA
{
   unsigned char Type; // тип возвращаемых данных
   unsigned char Share; // общий доступ к данным
   unsigned short Flags; // дополнительный флаг описания
   // общая область памяти
   union
   {
      struct // обработка портов ввода-вывода
      {
        PHYSICAL_ADDRESS Start; // базовый адрес
        unsigned long Length; // ширина порта
      } Port;
      struct // обработка номера прерывания
      {
        unsigned long Level; // уровень доступа
        unsigned long Number; // номер выделенного прерывания
        unsigned long Reserved; // резерв
      } IRQ;
      struct // обработка адресов в памяти
      {
        PHYSICAL_ADDRESS Start; // базовый адрес
        unsigned long Length; // ширина адресного пространства
      } Memory;
      struct
      {
        unsigned long Channel; // номер выделенного канала DMA
        unsigned long Port; // номер порта
        unsigned long Reserved; // резерв
      } DMA;
      struct // обработка номера шины
      {
        unsigned long Start; // базовый адрес шины
        unsigned long Length; // ширина адресного пространства
        unsigned long Reserved; // резерв
      } BusNumber;
   } u;
} IDDATA, *PIDDATA;
#pragma pack ()
typedef struct _DATA_LIST
{
   unsigned short Version; // номер версии
   unsigned short Revision; // дополнительный номер версии
```

```
   unsigned long Count; // полное количество данных об устройстве
   IDDATA Id[16]; // массив структур для получения данных
} DATA_LIST, *PDATA_LIST;
typedef struct _DATA_LIST_ALL
{
   DWORD Reserved; // резерв
   unsigned long Reserved2; // резерв
   DATA_LIST DataList; // структура DATA_LIST
} DATA_LIST_ALL, *PDATA_LIST_ALL;
typedef struct _DATA
{
   unsigned long Count; // количество найденных описателей устройства
   DATA_LIST_ALL All_List[2]; // массив структур DATA_LIST_ALL
} DATA, *PDATA;
// основная структура описателя устройства
typedef struct _DeviceParamNT
{
   char DeviceDesc[100]; // текстовое описание устройства
   char FriendlyName[80]; // дополнительное описание устройства
   char Class[50]; // имя класса устройства
   PHYSICAL_ADDRESS  Base_Memory[16]; // базовые адреса в памяти
   PHYSICAL_ADDRESS  End_Memory[16]; // завершающие адреса в памяти
   unsigned short TypeBaseMemory[16]; // тип адреса в памяти
   PHYSICAL_ADDRESS  Base_Port[16]; // базовый порт ввода-вывода
   PHYSICAL_ADDRESS  End_Port[16]; // завершающий порт ввода-вывода
   unsigned short TypeBasePort[16]; // тип порта ввода-вывода
   BYTE DMA_Channel[8]; // номер выделенного канала DMA
   unsigned short TypeDMA[8]; // тип канала DMA
   BYTE IRQ_Number[16]; // номер выделенного прерывания
   unsigned int BusNumber[8]; // номер шины для некоторых типов устройств
} DEVICEPARAMSNT, *PDEVICEPARAMSNT;
// определяем функцию поиска устройств в реестре
void GetDevicesConfig_NT ();
// массив структур DEVICEPARAMSNT для описания устройств
DEVICEPARAMSNT devices_NT[MAX_DEVICES];
// реализация функции
void GetDevicesConfig_NT ()
{
   // объявляем переменные
   HKEY phKey = NULL, phKey2 = NULL, phKey3 = NULL, phKey4 = NULL,
                       phKey5 = NULL; // дескрипторы разделов реестра
   // счетчики разделов и размеры значений параметров в реестре
   DWORD dwKey = 0, dwKey2 = 0, dwKey3 = 0, dwSize = 0, cbName = 256;
```

```
// счетчики данных
unsigned int uPort_index = 0, uIRQ_index = 0, uMemory_index = 0,
              uDMA_index = 0, uBus_index = 0;
// счетчик найденных устройств
unsigned int uIndex = 0;
// различные пути в реестре
char section_name[513];
char new_path[MAX_PATH];
char new_path2[MAX_PATH];
char new_path3[MAX_PATH];
char new_path4[MAX_PATH];
char new_path5[MAX_PATH];
// буфер для данных
char hard_key[150];
// структура для получения данных об устройстве
DATA cfg;
// обнуляем структуру перед использованием
for ( i = 0; i < 200; i++)
    ZeroMemory ( &devicesNT[i], sizeof ( DEVICEPARAMSNT));
// открываем раздел реестра для перечисления подразделов
if ( !RegOpenKeyEx ( HKEY_LOCAL_MACHINE, HKLM_ENUM_NT, 0, KEY_READ,
                      &phKey) == ERROR_SUCCESS)
      return;
while ( 1)
{
 // определяем размер значения
 cbName = 256;
 // перечисляем подразделы в открытом разделе
 if ( RegEnumKeyEx ( phKey, dwKey, section_name, &cbName, NULL, NULL,
                      NULL, NULL) == ERROR_NO_MORE_ITEMS)
    break; // выходим из цикла, если все подразделы получены
  // собираем путь для доступа к вложенному разделу
  strcpy ( new_path, "");
  strcpy ( new_path, HKLM_ENUM_NT);
  strcat ( new_path, "\\");
  strcat ( new_path, section_name);
  // открываем вложенный раздел
  if ( RegOpenKeyEx ( HKEY_LOCAL_MACHINE, new_path, 0, KEY_READ,
                       &phKey2) == ERROR_SUCCESS)
  {
    // сбрасываем счетчик
    dwKey2 = 0;
    while ( 1)
```

```
{
  // определяем размер значения
  cbName = 256;
  // перечисляем подразделы в открытом разделе
  if ( RegEnumKeyEx ( phKey2, dwKey2, new_path2, &cbName, NULL,
                      NULL, NULL, NULL) == ERROR_NO_MORE_ITEMS)
       break; // выходим из цикла, если все подразделы получены
  // собираем путь для доступа к вложенному разделу
  strcpy ( new_path3, "");
  strcpy ( new_path3, new_path);
  strcat ( new_path3, "\\");
  strcat ( new_path3, new_path2);
  // открываем вложенный раздел
  if ( RegOpenKeyEx ( HKEY_LOCAL_MACHINE, new_path3, 0, KEY_READ,
                      &phKey3) == ERROR_SUCCESS)
  {
    // сбрасываем счетчик
    dwKey3 = 0;
    while ( 1)
    {
      // определяем размер значения
      cbName = 256;
      if ( RegEnumKeyEx ( phKey3, dwKey3, new_path4, &cbName,
                 NULL, NULL, NULL, NULL) == ERROR_NO_MORE_ITEMS)
           break; // выходим из цикла
      // собираем путь для доступа к вложенному разделу
      strcpy ( new_path5, "");
      strcpy ( new_path5, new_path3);
      strcat ( new_path5, "\\");
      strcat ( new_path5, new_path4);
      // открываем вложенный раздел
      if(RegOpenKeyEx ( HKEY_LOCAL_MACHINE, new_path5, 0,
                        KEY_READ, &phKey4) == ERROR_SUCCESS)
      {
        dwSize = 150; // размер данных
        // получаем описание устройства
        RegQueryValueEx ( phKey4, "DeviceDesc", NULL, NULL,
                        ( LPBYTE) hard_key, &dwSize);
        // сохраняем в основную структуру
        strcpy ( devicesNT[uIndex].DeviceDesc, hard_key);
        strcpy ( hard_key, "");
        // получаем имя класса устройства
        dwSize = 50;
```

```
RegQueryValueEx ( phKey4, "Class", NULL, NULL,
                ( LPBYTE) hard_key, &dwSize);
strcpy ( devicesNT[uIndex].Class, hard_key);
strcpy ( hard_key, "");
// получаем дополнительное описание устройства
dwSize = 80;
RegQueryValueEx ( phKey4, "FriendlyName", NULL, NULL,
                ( LPBYTE) hard_key, &dwSize);
strcpy ( devicesNT[uIndex].FriendlyName, hard_key);
strcpy ( hard_key, "");
// собираем путь для доступа
// основным параметрам устройства
strcat ( new_path5, "\\Control"); // Windows 2000/XP/SR3
// strcat ( new_path5, "\\LogConf"); // Windows NT
// открываем раздел реестра
if ( RegOpenKeyEx ( HKEY_LOCAL_MACHINE, new_path5, 0,
                   KEY_READ, &phKey5) == ERROR_SUCCESS)
{
  dwSize = sizeof ( DATA); // определяем размер данных
  // обнуляем структуру данных
  ZeroMemory ( &cfg, sizeof ( DATA));
  // получаем данные из реестра
  RegQueryValueEx ( phKey5, "AllocConfig", NULL, NULL,
                  ( LPBYTE) &cfg, &dwSize);
  // определяем количество описателей устройства
  for ( int i = 0; i < cfg.Count; i++)
  {
    for ( int j = 0; j < cfg.All_List[i].DataList.Count;
          j++)
    {
      // определяем тип данных
      switch ( cfg.All_List[i].DataList.Id[j].Type)
      {
      case Port_Type: // порт ввода-вывода
       // получаем базовый порт
       devicesNT[uIndex].Base_Port[uPort_index] =
          cfg.All_List[i].DataList.Id[j].u.Port.Start;
       // получаем завершающий порт
       devicesNT[uIndex].End_Port[uPort_index].LowPart =
      cfg.All_List[i].DataList.Id[j].u.Port.Start.LowPart
      + cfg.All_List[i].DataList.Id[j].u.Port.Length - 1;
      // получаем тип порта ввода-вывода
      if ( cfg.All_List[i].DataList.Id[j].Flags &
           PORT_MEMORY)
```

```
      devicesNT[uIndex].TypeBasePort[uPort_index] =
         PORT_MEMORY;
    else if ( cfg.All_List[i].DataList.Id[j].Flags &
         PORT_IO)
      devicesNT[uIndex].TypeBasePort[uPort_index] =
         PORT_IO;
    // увеличиваем счетчик
    uPort_index++;
     break;
    case IRQ_Type: // номер прерывания
      devicesNT[uIndex].IRQ_Number[uIRQ_index] =
          cfg.All_List[i].DataList.Id[j].u.IRQ.Number;
      uIRQ_index++;
       break;
    case Memory_Type: // адреса в памяти
      // получаем базовый адрес
      devicesNT[uIndex].Base_Memory[uMemory_index] =
         cfg.All_List[i].DataList.Id[j].u.Memory.Start;
      // получаем завершающий адрес
     devicesNT[uIndex].End_Memory[uMemory_index].LowPart
= cfg.All_List[i].DataList.Id[j].u.Memory.Start.LowPart
+ cfg.All_List[i].DataList.Id[j].u.Memory.Length - 1;
      // определяем тип памяти
      if ( cfg.All_List[i].DataList.Id[j].Flags &
           MEMORY_READ_WRITE)
        devicesNT[uIndex].TypeBaseMemory[uPort_index] =
           MEMORY_READ_WRITE;
      else if ( cfg.All_List[i].DataList.Id[j].Flags &
           MEMORY_READ_ONLY)
        devicesNT[uIndex].TypeBaseMemory[uPort_index] =
           MEMORY_READ_ONLY;
        devicesNT[uIndex].TypeBaseMemory[uPort_index] =
           MEMORY_READ_ONLY;
        devicesNT[uIndex].TypeBaseMemory[uPort_index] =
           MEMORY_WRITE_ONLY;
       uMemory_index++;
       break;
      case DMA_Type: // номер канала DMA
       // получаем номер канала
       devicesNT[uIndex].DMA_Channel[uDMA_index] =
           cfg.All_List[i].DataList.Id[j].u.DMA.Channel;
       // определяем тип канала
       if ( cfg.All_List[i].DataList.Id[j].Flags &
            DMA_8)
```

```
                        devicesNT[uIndex].TypeDMA[uPort_index] =  DMA_8;
                       else if ( cfg.All_List[i].DataList.Id[j].Flags &
                            DMA_16)
                       devicesNT[uIndex].TypeDMA[uPort_index] =  DMA_16;
                       else if ( cfg.All_List[i].DataList.Id[j].Flags &
                            DMA_32)
                       devicesNT[uIndex].TypeDMA[uPort_index] =  DMA_32;
                       uDMA_index++;
                       break;
                      case BusNumber_Type: // тип шины
                       devicesNT[uIndex].BusNumber[uBus_index] =
                       cfg.All_List[i].DataList.Id[j].u.BusNumber.Start;
                       break;
                    }
                  }
                }
                // закрываем раздел и обнуляем счетчики
                if ( phKey5) RegCloseKey ( phKey5);
                uPort_index = 0;
                uIRQ_index = 0;
                uMemory_index = 0;
                uDMA_index = 0;
                uBus_index = 0;
              }
              // закрываем вложенный раздел
              if ( phKey4) RegCloseKey ( phKey4);
              // увеличиваем счетчик найденных устройств
              uIndex++;
            }
            // увеличиваем счетчик для поиска следующего подраздела
            dwKey3++;
          }
          // закрываем вложенный раздел
          if ( phKey3) RegCloseKey ( phKey3);
        }
        // увеличиваем счетчик для поиска следующего подраздела
        dwKey2++;
      }
      // закрываем вложенный раздел
      if ( phKey2) RegCloseKey ( phKey2);
    }
   // увеличиваем счетчик для получения следующего подраздела
   dwKey++;
  }
```

```
    // закрываем корневой раздел и выходим из функции
    if ( phKey) RegCloseKey ( phKey);
}
```

Как видите, основные принципы получения информации об установленном оборудовании в профессиональной системе Windows не сильно отличаются от пользовательского варианта. Изменилась только структура для хранения данных и названия разделов в реестре. Замечу только, что ключевой раздел, в котором хранятся описатели устройств, может иметь одно из двух возможных названий: **LogConf** для Windows NT 4.0 и более ранних версий или **Control** для Windows 2000/XP/SR3.

23.2. Самоликвидация исполняемого файла

Необходимость в удалении исполняемым файлом (расширение exe) самого себя возникает, в первую очередь, при разработке инсталляционных приложений. Можно конечно переложить всю работу на плечи потенциального пользователя, но вряд ли ваша программа от этого выиграет. Как бы там ни было, я расскажу о нескольких вариантах удаления EXE-файла, а выбор наиболее приемлемого способа останется за вами.

Первый предлагаемый вариант будет работать только в операционных системах Windows 9*x*/ME. Он позволит удалить не только сам файл программы, но и папку, если таковая имеется. Однако следует помнить, что удалить можно только пустую папку, поэтому перед вызовом функции удаления EXE-файла не забывайте очищать текущий каталог от вложенных папок и файлов. Пример функции, демонстрирующий первый способ самоуничтожения исполняемого файла, представлен в листинге 23.3.

Листинг 23.3. Первый вариант удаления исполняемого файла

```
#include <windows.h>
// объявляем переменные
// указатель на функцию FreeLibrary
typedef BOOL ( FAR PASCAL* PFNFREELIBRARY) ( HINSTANCE);
// указатель на функцию DeleteFile
typedef BOOL ( FAR PASCAL* PFNDELETEFILE) ( LPCTSTR);
// указатель на функцию RemoveDirectory
typedef BOOL ( FAR PASCAL* PFNREMOVEDIRECTORY) ( LPCTSTR);
// указатель на функцию ExitProcess
typedef void ( FAR PASCAL* PFNEXITPROCESS) ( DWORD);
// структура для хранения основных параметров
typedef struct
```

```
{
   HINSTANCE hinst; // дескриптор исполняемого файла
   DWORD dwExitCode; // код завершения процесса
   char szPathFile[MAX_PATH]; // путь к исполняемому файлу
   char szPathFolder[MAX_PATH]; // путь к папке
   // указатели на системные функции
   PFNFREELIBRARY pfnFreeLibrary;
   PFNDELETEFILE pfnDeleteFile;
   PFNREMOVEDIRECTORY pfnRemoveDirectory;
   PFNEXITPROCESS pfnExitProcess;
} SELFDELEXE, *PSELFDELEXE;
// определяем дальний указатель
typedef void ( FAR PASCAL* CALL_EXE} ( PSELFDELEXE);
// функция удаления EXE-файла
void SelfExeDelete_9X ( BOOL bIsFolder);
// функция загрузки данных для удаления
static void LoadData ( PSELFDELEXE pExe);
// функция маркировки конечной точки в стеке
static void EndDataStack ();
// реализация функций
static void LoadData ( PSELFDELEXE pExe)
{
   // выгружаем файл из текущего процесса
   pExe->pfnFreeLibrary ( pExe->hinst);
   // удаляем его
   pExe->pfnDeleteFile ( pExe->szPathFile);
   // удаляем папку, если она пустая
   if ( pExe->pfnRemoveDirectory)
       pExe->pfnRemoveDirectory ( pExe->szPathFolder);
   // завершаем текущий процесс
   pExe->pfnExitProcess ( pExe->dwExitCode);
}
static void EndDataStack ()
{
   // просто отмечаем в стеке завершение данных
}
void  SelfExeDelete_9X ( BOOL bIsFolder)
{
   SELFDELEXE exe; // структура с данными
   DWORD dwDataSize = 0; // размер данных
   DWORD dwExitCode = 0xDADADADA; // код завершения
   HINSTANCE hKernel = NULL; // дескриптор модуля
   HANDLE hMemory = NULL; // дескриптор памяти
   CALL_EXE pfnExe = NULL; // дальний указатель
```

```
// обнуляем структуру перед использованием
ZeroMemory ( &exe, sizeof ( SELFDELEXE));
// определяем размер данных
dwDataSize = ( ( LPBYTE) ( DWORD) EndDataStack -
               ( LPBYTE) ( DWORD) LoadData);
// получаем дескриптор загруженного системного модуля
hKernel = GetModuleHandle ( "KERNEL32");
// в случае ошибки выходим из функции
if ( hKernel == NULL) return;
// получаем дескриптор памяти для текущего процесса
hMemory = GetProcessHeap ();
// в случае ошибки выходим из функции
if ( hMemory == NULL) return;
// выделяем свободную память для данных
pfnExe = HeapAlloc ( hMemory, HEAP_ZERO_MEMORY, dwDataSize);
// записываем в выделенную память наши данные
memcpy ( pfnExe, LoadData, dwDataSize);
// заполняем пустые данные содержанием
exe.hinst = GetModuleHandle ( NULL); // дескриптор EXE-файла
// указатель на функцию FreeLibrary
exe.pfnFreeLibrary = ( PFNFREELIBRARY) GetProcAddress ( hKernel,
                                        "FreeLibrary");
// указатель на функцию DeleteFile для ANSI
exe.pfnDeleteFile = ( PFNDELETEFILE) GetProcAddress ( hKernel,
                                      "DeleteFileA");
// получаем имя и полный путь для EXE-файла
GetModuleFileName ( exe.hinst, exe.szPathFile, MAX_PATH);
// если необходимо удалить папку
if ( bIsFolder)
{
  // указатель на функцию RemoveDirectory для ANSI
  exe.pfnRemoveDirectory = ( PFNREMOVEDIRECTORY)
                 GetProcAddress ( hKernel, "RemoveDirectoryA");
  // удаляем имя файла из пути
  strcpy ( exe.szPathFolder, exe.szPathFile);
  *strchr ( exe.szPathFolder, '\\') = 0;
}
else // иначе записываем нулевое значение
  exe.pfnRemoveDirectory = NULL;
// указатель на функцию ExitProcess
exe.pfnExitProcess = ( PFNEXITPROCESS) GetProcAddress ( hKernel,
                                        "ExitProcess");
// записываем код завершения
exe.dwExitCode = dwExitCode;
```

```
   // передаем управление на область памяти, в которой находятся данные
   // для самоликвидации EXE-файла
   pfnExe ( &exe);
}
// пример использования функции SelfDelete_9X
int CALLBACK WinMain ( HINSTANCE hInstance, HINSTANCE hPrewInstance,
                       LPSTR lpCmdLine, int nCmdShow)
{
   // удаляем сами себя
   SelfDelete_9X ( FALSE);
   return 0;
}
```

Принцип работы функции `SelfExeDelete_9X` основан на добавлении в свободную память выполняемого процесса специального кода. Данный код становится независимой частью процесса и позволяет выполнять любые действия по отношению к остальным загруженным модулям, в том числе и EXE-файлу.

Второй рассматриваемый вариант удаления файла можно применять во всех версиях Windows, но для завершения операции потребуется перезагрузка системы. В папке с установленной системой Windows присутствует программа Wininit.exe. Она во время каждого запуска проверяет папку на наличие файла Wininit.ini. Если такой файл существует, программа обрабатывает его и по завершению переименовывает в Wininit.bak. Сам файл настройки позволяет переименовать или удалить указанные файловые объекты, размещенные в его разделе **Rename**. Пример функции показан в листинге 23.4.

Листинг 23.4. Второй вариант удаления исполняемого файла

```
#include <windows.h>
// функция удаления EXE-файла
void  SelfExeDelete ( BOOL bIsFolder);
// пишем реализацию функции удаления
void  SelfExeDelete ( BOOL bIsFolder)
{
   // объявляем переменные
   char szPathEXE[MAX_PATH]; // путь к файлу EXE
   char szLineINI[1023]; // строка описания в файле WinInit.ini
   char szWindows[MAX_PATH]; // путь к папке Windows ( WinNT)
   char szSection[] = "[Rename]\r\n"; // указатель раздела в Wininit.ini
   char* pRename = NULL;
   char* pWinINIT = NULL;
   char* ptr = NULL;
```

```
int iLine_Size = 0, iSection_Size = 0;
HANDLE hFile = NULL, hMapFile = NULL;
DWORD dwFileSize = 0, dwPos = 0;
// получаем имя и полный путь к файлу EXE
GetModuleFileName ( NULL, szPathEXE, MAX_PATH);
// пытаемся пометить наш файл для удаления после перезагрузки системы
if ( !MoveFileEx ( szPathEXE, NULL, MOVEFILE_DELAY_UNTIL_REBOOT))
{
  // если ошибка, выполняем операцию вручную
  // создаем строку для записи в файл Wininit.ini
  iLine_Size = sprintf ( szLineINI, "NUL=%hs\r\n", szPathEXE);
  // вычисляем размер заголовка
  iSection_Size = sizeof ( szSection) - 1;
  // получаем путь к папке Windows
  GetWindowsDirectory ( szWindows, MAX_PATH);
  // добавляем имя файла настройки
  strcat ( szWindows, "\\Wininit.ini");
  // открываем или создаем новый файл Wininit.ini
  hFile = CreateFile ( szWindows, GENERIC_READ | GENERIC_WRITE, 0,
                       NULL, OPEN_ALWAYS, FILE_FLAG_SEQUENTIAL_SCAN |
                       FILE_ATTRIBUTE_NORMAL, NULL);
  // если не удалось создать файл, выходим из функции
  if ( hFile == INVALID_HANDLE_VALUE) return;
  // определяем размер файла
  dwFileSize = GetFileSize( hFile, NULL);
  // создаем неименованный объект отображения нашего файла
  hMapFile = CreateFileMapping ( hFile, NULL, PAGE_READWRITE, 0,
                   dwFileSize + iLine_Size + iSection_Size, NULL);
  // если ошибок нет, работаем с новым объектом
  if ( hMapFile)
  {
    // получаем объект отображения файла
    pWinINIT = ( char*) MapViewOfFile ( hMapFile, FILE_MAP_WRITE, 0,
                                         0, 0);
    // если нет ошибок, ищем в файле нужный раздел
    if ( pWinINIT)
    {
      pRename = strstr ( pWinINIT, szSection);
      if ( pRename) // если раздел [Rename] найден
      {
        // вычисляем конец данных для добавления своих
        ptr = strchr ( pRename, '\n');
        ptr++;
```

```
          memmove ( ptr + iLine_Size, ptr,
                    pWinINIT + dwFileSize - ptr);
          // запоминаем позицию указателя в файле
          dwPos = ptr - pWinINIT;
        }
        else // если раздел отсутствует
        {
          // записываем в файл имя раздела [Rename]
          dwFileSize += wsprintf ( &pWinINIT[dwFileSize], "%s",
                                   szSection);
           // запоминаем позицию указателя в файле
          dwPos = dwFileSize;
        }
        // записываем в файл свои данные
        memcpy ( &pWinINIT[dwPos], szLineINI, iLine_Size);
        // освобождаем объект отображения файла
        UnmapViewOfFile ( pWinINIT);
        // получаем новый размер файла
        dwFileSize += iLine_Size;
      }
      // закрываем файловый объект
      CloseHandle ( hMapFile);
    }
    // закрываем файл
    SetFilePointer ( hFile, dwFileSize, NULL, FILE_BEGIN);
    SetEndOfFile ( hFile);
    CloseHandle ( hFile);
  }
}
```

Пример использования функции `SelfExeDelete` я не привожу, поскольку он ничем не отличается от предыдущего. При желании после несложных исправлений данную функцию можно применять для любых типов файлов.

И последний способ удаления EXE-файла, который я хочу вам представить, заключается в создании старого доброго пакетного файла BAT. Работать этот способ будет во всех версиях Windows, поддерживающих BAT-файлы. В листинге 23.5 показан пример реализации такого способа.

Листинг 23.5. Третий вариант удаления исполняемого файла

```
#include <Windows.h>
// функция удаления EXE-файла
void  SelfExeDelete ( BOOL bIsFolder);
```

```
// пишем реализацию функции удаления
void  SelfExeDelete ( BOOL bIsFolder)
{
   // объявляем переменные
   HANDLE hFile = NULL; // дескриптор файла
   PROCESS_INFORMATION proc_info; // информация о процессе
   STARTUPINFO info; // информация о начальной загрузке программы
   DWORD dwBytesWrite = 0; // число записанных байтов
   char szBuffer[1000]; // буфер для данных
   char szExe[MAX_PATH]; // путь и имя EXE-файла
   char szPath[MAX_PATH]; // путь к EXE-файлу
   char BAT[] = "\\TmpDelExe.bat"; // имя временного пакетного файла
   // обнуляем структуры
   ZeroMemory ( &proc_info, sizeof ( PROCESS_INFORMATION));
   ZeroMemory ( &info, sizeof ( STARTUPINFO));
   // создаем новый файл для записи
   hFile = CreateFile ( BAT, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN, NULL);
   // если ошибка, выходим из функции
   if ( hFile != INVALID_HANDLE_VALUE)
   {
     // получаем имя и путь к EXE-файлу
     GetModuleFileName ( NULL, szExe, MAX_PATH);
     // выделяем только путь
     strcpy ( szPath, szExe);
     *strchr ( szPath, '\\') = 0;
     // форматируем данные для BAT-файла
     sprintf(szBuffer,
         ":Repeat\r\ndel \"%s\"\r\nif exist \"%s\" goto"
         " Repeat\r\nrmdir \"%s\"\r\ndel \"%s\"\r\n",
         szExe, szExe, szPath, BAT);
     // записываем данные в файл и закрываем его
     WriteFile ( hFile, szBuffer, strlen ( szBuffer) * sizeof ( char),
               &dwBytesWrite, NULL);
     CloseHandle ( hFile);
     // заполняем структуру на запуск
     info.cb = sizeof ( info); // размер структуры
     info.dwFlags = STARTF_USESHOWWINDOW; // устанавливаем обработку окна
     info.wShowWindow = SW_HIDE; // скрытое окно
     // создаем новый процесс
     if ( CreateProcess ( NULL, BAT, NULL, NULL, FALSE, CREATE_SUSPENDED
              | IDLE_PRIORITY_CLASS, NULL, "\\", &info, &proc_info))
     {
       // устанавливаем для созданного потока самый низкий приоритет
       SetThreadPriority ( proc_info.hThread, THREAD_PRIORITY_IDLE);
```

```
        // устанавливаем наивысший приоритет для главного потока EXE-файла
        SetThreadPriority ( GetCurrentThread (),
                            THREAD_PRIORITY_TIME_CRITICAL);
        // устанавливаем высокий приоритет для текущего потока
        SetPriorityClass ( GetCurrentProcess (), HIGH_PRIORITY_CLASS);
        // закрываем дескриптор процесса
        CloseHandle ( proc_info.hProcess);
        // приостанавливаем выполнение потока
        ResumeThread ( proc_info.hThread);
        // закрываем дескриптор потока
        CloseHandle ( proc_info.hThread);
      }
   }
}
```

Итак, сначала мы создали новый пакетный файл и записали в него стандартные команды DOS для удаления файла и родительской папки. После этого на основе BAT-файла открыли новый процесс с помощью функции `CreateProcess` и запустили его на выполнение. И наконец, изменив приоритеты текущего и вновь созданного процесса, закрыли дескрипторы созданного процесса и его основного потока. Система запустила на выполнение пакетный файл TmpDelExe.bat, удалив тем самым наш EXE-файл. Чтобы лучше понять работу функций DOS, ознакомьтесь с соответствующим руководством.

23.3. Создание прозрачных окон

Эта интересная возможность, довольно легко реализуемая в Windows 2000/XP/SR3, к сожалению, не поддерживается в пользовательских системах Windows 9*x*/ME. Чтобы восполнить этот пробел, напишем собственный код, позволяющий превратить любое окно (в том числе и элементы управления) в прозрачную поверхность. Для этого нам понадобится написать всего три несложных функции. Создайте новый проект и добавьте стандартный код регистрации класса окна. Добавьте оконную функцию обработки сообщений Windows и определите следующие сообщения: `WM_CREATE`, `WM_PAINT`, `WM_SIZE` и `WM_WINDOWPOSCHANGED`. Обработайте эти сообщения так, как показано в листинге 23.6.

Листинг 23.6. Создание прозрачного окна

```
#include "stdafx.h"
// объявляем глобальные переменные
HBITMAP hDesktop = NULL, HBITMAP hDesktop2 = NULL, hScreen = NULL;
RECT rect, rect2, rect3;
```

```
// уровень прозрачности
int iLevel = 100;
// копирование текущего содержимого экрана
HBITMAP  SaveScreen ( LPRECT lpRect);
// перерисовка прозрачного окна
void UpdatePaintTrans ( HWND hWnd);
// создание эффекта прозрачности
void Transparency ( HDC dcIn, HDC dcOut, int left_in, int top_in,
   int right_in, int bottom_in,  int left_out, int top_out,
    int right_out, int bottom_out);
// оконная функция для обработки сообщений Windows
LRESULT CALLBACK WndProc ( HWND hWnd, UINT msg, WPARAM wParam,
                           LPARAM lParam);
// реализация функций
HBITMAP SaveScreen ( LPRECT lpRect)
{
   HBITMAP hBmp = NULL, hTmpBmp = NULL; // дескрипторы изображений
   HDC inDC = NULL, memDC = NULL; // дескрипторы для контекстов
   int iWidth = 0, iHeight = 0; // размеры изображения
   int x_res = 0, y_res = 0; // текущее разрешение экрана
   RECT r; // хранение размеров прямоугольника
   // проверяем корректность размеров прямоугольника
   if ( IsRectEmpty ( lpRect))
      return NULL; // в случае недопустимых координат, выходим
   // создаем контекст для стандартного устройства отображения
   inDC = CreateDC ( "DISPLAY", NULL, NULL, NULL);
   // создаем в памяти контекст, совместимый с созданным ранее
   memDC = CreateCompatibleDC ( inDC);
   // определяем текущее разрешение экрана в пикселах
   x_res = GetDeviceCaps ( inDC, HORZRES); // по горизонтали
   y_res = GetDeviceCaps ( inDC, VERTRES); // по вертикали
   // получаем размеры прямоугольника
   r.left = lpRect->left;
   r.top = lpRect->top;
   r.right = lpRect->right;
   r.bottom = lpRect->bottom;
   // вычисляем размеры прямоугольника
   iWidth = r.right - r.left;
   iHeight = r.bottom - r.top;
   // создаем изображение с заданными размерами
   hBmp = CreateCompatibleBitmap ( inDC, iWidth, iHeight);
   // копируем изображение в контекст памяти
   hTmpBmp = ( HBITMAP) SelectObject ( memDC, hBmp);
```

```
    // копируем изображение из памяти на экран
    BitBlt ( memDC, 0, 0, iWidth, iHeight, inDC, r.left, r.top, SRCCOPY);
    // восстанавливаем прежнее изображение для контекста
    hBmp = ( HBITMAP) SelectObject ( memDC, hTmpBmp);
    // освобождаем ресурсы системы
    DeleteDC ( memDC);
    DeleteDC ( inDC);
    // возвращаем дескриптор изображения
    return hBmp;
}
void UpdatePaintTrans ( HWND hWnd)
{
    PAINTSTRUCT pts; // структура для рисования
    HDC hDisplay = NULL, memDC = NULL, bmpDC = NULL, tmpDC = NULL;
    RECT r, r2, r3;
    HBITMAP memBmp = NULL, oldBmp = NULL, tmpBmp = NULL, tmpBmp2 = NULL,
            tmpBmp3 = NULL, tmpBmp4 = NULL;
    // получаем контекст для рисования
    HDC hDC = BeginPaint ( hWnd, &pts);
    // получаем текущие размеры окна
    GetWindowRect ( hWnd, &r);
    // сохраняем размеры прямоугольника для дальнейшего использования
    r2.bottom = r.bottom - r.top;
    r2.right  = r.right  - r.left;
    // создаем контекст для стандартного устройства отображения
    hDisplay = CreateDC ( "DISPLAY", NULL, NULL, NULL);
    // создаем в памяти контекст, совместимый с созданным ранее
    memDC = CreateCompatibleDC ( hDisplay);
    // создаем временный контекст для обработки изображения
    tmpDC = CreateCompatibleDC ( hDC);
    // определяем координаты пересекающихся прямоугольников
    if ( IntersectRect ( &r3, &r, &rect3))
    {
      // создаем в памяти совместимый контекст
      memDC = CreateCompatibleDC ( hDisplay);
      // копируем изображение в контекст
      memBmp = ( HBITMAP) SelectObject ( memDC, hDesktop);
      // копируем изображение в контекст
      oldBmp = ( HBITMAP) SelectObject ( tmpDC, hDesktop2);
      // копируем изображение из одного контекста в другой
      BitBlt ( memDC, r3.left, r3.top, r3.right, r3.bottom, tmpDC,
               r3.left, r3.top, SRCCOPY);
      // восстанавливаем прежнее изображение
      SelectObject ( tmpDC, oldBmp);
```

```
    // копируем изображение в контекст памяти
    hDesktop = ( HBITMAP) SelectObject ( memDC, memBmp);
  }
  // копируем размеры прямоугольника
  rect3 = r;
  // копируем изображение в контекст памяти
  tmpBmp = ( HBITMAP) SelectObject ( memDC, hDesktop);
  // создаем битовое изображение
  tmpBmp2 = CreateBitmap ( r2.right, r2.bottom, 1,
                              GetDeviceCaps ( hDC, BITSPIXEL), NULL);
  // копируем изображение в контекст
  tmpBmp3  = ( HBITMAP) SelectObject ( tmpDC, tmpBmp2);
  // копируем изображение из одного контекста в другой
  BitBlt ( tmpDC, 0, 0, r2.right, r2.bottom, memDC, r.left, r.top,
          SRCCOPY);
  // выбираем изображение в контекст памяти
  hDesktop2 = ( HBITMAP) SelectObject ( memDC, tmpBmp);
  // освобождаем контекст
  DeleteDC ( memDC);
  // создаем совместимый контекст
  bmpDC = CreateCompatibleDC ( hDC);
  // выбираем изображение в контекст
  tmpBmp4  = ( HBITMAP) SelectObject ( bmpDC, hScreen);
  // создаем эффект прозрачности для указанного изображения
  Transparency ( tmpDC, bmpDC, 0, 0, r2.right, r2.bottom, 0, 0,
               r2.right, r2.bottom);
  // копируем изображение из одного контекста в другой
  BitBlt ( hDisplay, r.left, r.top, r2.right, r2.bottom, tmpDC, 0, 0,
          SRCCOPY);
  // восстанавливаем прежнее изображение
  SelectObject ( tmpDC, tmpBmp2);
  // освобождаем контекст
  DeleteDC ( tmpDC);
  // восстанавливаем прежнее изображение
  SelectObject ( bmpDC, tmpBmp4);
  // освобождаем контекст
  DeleteDC ( bmpDC);
  // завершаем рисование
  EndPaint ( hWnd, &pts);
  DeleteDC(hDisplay); // освобождаем контекст
  // сохраняем текущее изображение экрана
  hDesktop = SaveScreen ( &rect);
}
```

```
void Transparency ( HDC dcIn, HDC dcOut, int left_in, int top_in,
 int right_in, int bottom_in, int left_out, int top_out, int right_out,
 int bottom_out)
{
   // структура для хранения параметром битового изображения
   BITMAPINFOHEADER bmp;
   BYTE* pInBits = NULL;
   BYTE* pOutBits = NULL;
   HBITMAP hInBmp = NULL, hOutBmp = NULL, hTmpBmp = NULL;
   HDC dc = NULL;
   // обнуляем структуру
   ZeroMemory ( &bmp, sizeof ( BITMAPINFOHEADER));
   // заполняем структуру данными
   bmp.biSize = sizeof ( BITMAPINFOHEADER); // размер структуры
   bmp.biWidth = right_in; // ширина изображения
   bmp.biHeight = bottom_in; // высота изображения
   bmp.biPlanes = 1; // число плоскостей всегда равно 1
   bmp.biBitCount = 32; // число битов для описания одного пиксела
   bmp.biCompression = BI_RGB; // без сжатия данных
   // создаем целевое растровое изображение
   hInBmp = CreateDIBSection ( dcOut, ( BITMAPINFO*) &bmp,
                     DIB_RGB_COLORS, (void**) &pInBits, NULL, 0);
   // создаем исходное растровое изображение
   hOutBmp = CreateDIBSection ( dcIn, ( BITMAPINFO*) &bmp,
                     DIB_RGB_COLORS, ( void**) &pOutBits, NULL, 0);
   // создаем контекст, совместимый с текущим экраном
   dc = CreateCompatibleDC ( NULL);
   // копируем изображение в контекст
   hTmpBmp = ( HBITMAP) SelectObject ( dc, hInBmp);
   // копируем целевое изображение и подгоняем размеры
   StretchBlt ( dc, 0, 0, right_in, bottom_in, dcOut, left_out, top_out,
             right_out, bottom_out, SRCCOPY);
   // выбираем изображение в контекст
   SelectObject ( dc, hOutBmp);
   // копируем исходное изображение и подгоняем размеры
   StretchBlt ( dc, 0, 0, right_in, bottom_in, dcIn, left_in, top_in,
             right_in, bottom_in, SRCCOPY);
   // выбираем изображение в контекст
   SelectObject ( dc, hTmpBmp);
   DeleteDC ( dc);
   // обрабатываем каждый байт изображения для получения прозрачности
   for ( int i = 0; i < bottom_in; ++i) // по вертикали
    {
     LPBYTE pInRGB = ( LPBYTE) &( ( DWORD*) pInBits)[i * right_in];
```

```
      LPBYTE pOutRGB = ( LPBYTE) & ( ( DWORD*) pOutBits)[i * right_in];
      for ( int j = 0; j < right_in; ++j) // по горизонтали
      {
       // вычисляем новое значение каждой составляющей цвета
       pInRGB[0] = ( pOutRGB[0] * ( 255 - iLevel) + pInRGB[0] *
                   iLevel) >> 8;
       pInRGB[1] = ( pOutRGB[1] * ( 255 - iLevel) + pInRGB[1] *
                     iLevel) >> 8;
       pInRGB[2] = ( pOutRGB[2] * ( 255 - iLevel) + pInRGB[2] *
                     iLevel) >> 8;
       pInRGB += 4;
       pOutRGB += 4;
      }
     }
    // создаем контекст, совместимый с текущим экраном
    dc = CreateCompatibleDC ( NULL);
    // выбираем обработанное изображение
    hTmpBmp = ( HBITMAP) SelectObject ( dc, hInBmp);
    // копируем изображение из одного контекста в другой
    BitBlt ( dcIn, left_in, top_in, right_in, bottom_in, dc, 0, 0,
             SRCCOPY);
    // освобождаем системные ресурсы
    DeleteObject ( hInBmp);
    DeleteObject ( hOutBmp);
    DeleteDC ( dc);
}
// основная оконная функция программы
LRESULT CALLBACK WndProc ( HWND hWnd, UINT msg, WPARAM wParam,
                           LPARAM lParam)
(
    RECT r; // хранение координат прямоугольника
    // выполняем обработку сообщений
    switch ( msg)
    {
     case WM_CREATE:
     {
       // получаем размеры клиентской области окна
       GetClientRect ( GetDesktopWindow (), &rect);
       // копируем в виде растрового изображения в память
       hDesktop = SaveScreen ( &rect);
     }
     break;
```

```
case WM_PAINT:
{
   // перерисовываем окно
   UpdatePaintTrans ( hWnd);
}
 break;
case WM_SIZE:
{
   // получаем размеры окна
   GetWindowRect ( hWnd, &r);
   // обновляем содержимое окна
   InvalidateRect ( hWnd, &r, true);
   // копируем в виде растрового изображения в память
   hScreen = SaveScreen ( &r);
}
 break;
case WM_WINDOWPOSCHANGED:
{
  LPWINDOWPOS pos;
  // выделяем указатель на структуру LPWINDOWPOS
  pos = ( LPWINDOWPOS) lParam;
  // проверяем текущие размеры окна
  if ( ( pos->cx != rect2.right) ||  ( pos->cy != rect2.bottom))
  {
    // получаем размеры окна
    GetWindowRect ( hWnd, &r);
    // обновляем содержимое окна
    InvalidateRect ( hWnd, &r, true);
    // копируем в виде растрового изображения в память
    hScreen = SaveScreen ( &r);
  }
  if ( ( pos->x != rect2.left) || ( pos->y != rect2.top))
    PostMessage ( hWnd, WM_PAINT, 0, 0); // перерисовка окна
  // сохраняем размеры окна
  rect2.left = pos->x;
  rect2.right = pos->cx;
  rect2.top = pos->y;
  rect2.bottom = pos->cy;
}
 break;
// завершение работы программы
case WM_DESTROY:
   PostQuitMessage ( 0);
 break;
```

```
   default:
    return DefWindowProc ( hWnd, msg, wParam, lParam);
   }
   return 0;
}
```

Представленный код позволяет отображать главное окно программы полупрозрачным. Уровень эффекта можно изменять с помощью значения переменной `iLevel`. Для создания прозрачных окон в Windows 2000/XP/SR3 достаточно воспользоваться библиотечной функцией `SetLayeredWindowAttributes`. Кроме того, не требуется дополнительной обработки системных сообщений, поскольку все это система выполняет самостоятельно. Изменять уровень прозрачности можно в любой момент выполнения программы. Те, кто использует Visual Studio .NET, могут вызывать функцию непосредственно, а в VC++ 6.0 имеет смысл вручную определить требуемые константы и применить динамический способ обращения к системной библиотеке. Связано это в первую очередь с устаревшими версиями файлов определений, входящих в состав оболочки. Пример реализации прозрачного окна представлен в листинге 23.7.

Листинг 23.7. Создание прозрачного окна в Windows 2000

```
#include "stdafx.h"
// объявляем глобальные переменные
#define LWA_ALPHA  0x00000002 // используем эффект прозрачности
#define WS_EX_LAYERED  0x00080000 // расширенный стиль окна
// уровень прозрачности
BYTE byLevel = 100; // от 0 ( прозрачно) до 255 ( не прозрачно)
HMODULE hUser = NULL; // дескриптор библиотеки User32.dll
// указатель на функцию SetLayeredWindowAttributes
typedef BOOL ( WINAPI* PFNSETLAYEREDWINATTR) ( HWND hwnd,
                   COLORREF crKey, BYTE bAlpha, DWORD dwFlags);
// функция для добавления эффекта прозрачности
void Transparency ( HWND hWnd, BYTE byLevel);
// реализация функции
void Transparency ( HWND hWnd, BYTE byLevel)
{
   // указатель на функцию SetLayeredWindowAttributes
   PFNSETLAYEREDWINATTR pfnSetLayeredWindowAttributes;
   // загружаем библиотеку User32.dll
   hUser = GetModuleHandle ( "USER32");
   if ( hUser == NULL) return; //выходим в случае ошибки
   // добавляем к окну стиль для поддержки прозрачности
   SetWindowLong ( hWnd, GWL_EXSTYLE, WS_EX_LAYERED);
```

```
    // получаем указатель на функцию SetLayeredWindowAttributes
    pfnSetLayeredWindowAttributes =
                 ( PFNSETLAYEREDWINATTR) GetProcAddress ( hUser,
                   "SetLayeredWindowAttributes");
    // если доступ к функции получен
    if ( pfnSetLayeredWindowAttributes != NULL)
    {
      // устанавливаем уровень прозрачности для главного окна
      pfnSetLayeredWindowAttributes ( hWnd, NULL, byLevel, LWA_ALPHA);
    }
}
```

Думаю, пример достаточно простой и не требует специальных пояснений. Добавьте функцию `Transparency` в обработку сообщения `WM_CREATE` и не забудьте по завершению программы освободить библиотеку с помощью `FreeLibrary`.

23.4. Определение устройств USB

В последнее время увеличился интерес разработчиков к программированию устройств с интерфейсом USB. Это объясняется тем, что появилось огромное количество всевозможного оборудования: звуковые карты, винчестеры, приводы CD-ROM и CD-RW, сканеры и принтеры. К сожалению, свободно распространяемая документация очень скудно описывает методы работы с данными устройствами. Я постараюсь отчасти восполнить этот пробел и расскажу о базовых приемах, позволяющих получить различную полезную информацию о подключенных устройствах USB.

В первую очередь, нам понадобятся файлы определений (в частности hidsdi.h), входящие в состав пакета для разработки драйверов (DDK — Driver Developers Kits). Данный пакет можно свободно скачать с официального сайта фирмы Microsoft (**www.microsoft.ru**). Кроме того, необходимо будет добавить в опциях компоновщика ссылку на библиотеки Hid.lib и Setupapi.lib. Чтобы упростить код, вынесем в отдельный файл необходимые для поддержки USB константы и структуры, и назовем его usbdefs.h. Полное описание этого файла представлено в листинге 23.8.

Листинг 23.8. Файл поддержки USB-устройств usbdefs.h

```
// подключаем файл определений из пакета DDK
extern "C"
{
   #include "hidsdi.h"
}
```

```
// базовые коды ввода-вывода
#include <winioctl.h>
// максимальная длина строки описания
#define MAXIMUM_USB_STRING_LENGTH 255
// типы дескрипторов
#define USB_CONFIGURATION_DESCRIPTOR_TYPE   0x02
#define USB_STRING_DESCRIPTOR_TYPE          0x03
#define USB_INTERFACE_DESCRIPTOR_TYPE       0x04
// типы классов для устройств USB
#define USB_DEVICE_CLASS_RESERVED           0x00
#define USB_DEVICE_CLASS_AUDIO              0x01
#define USB_DEVICE_CLASS_COMMUNICATIONS     0x02
#define USB_DEVICE_CLASS_HUMAN_INTERFACE    0x03
#define USB_DEVICE_CLASS_MONITOR            0x04
#define USB_DEVICE_CLASS_PHYSICAL_INTERFACE 0x05
#define USB_DEVICE_CLASS_POWER              0x06
#define USB_DEVICE_CLASS_PRINTER            0x07
#define USB_DEVICE_CLASS_STORAGE            0x08
#define USB_DEVICE_CLASS_HUB                0x09
#define USB_DEVICE_CLASS_VENDOR_SPECIFIC    0xFF
// коды ввода-вывода для функции DeviceIoControl
#define FILE_DEVICE_USB       FILE_DEVICE_UNKNOWN
#define USB_IOCTL_INDEX       0x00ff
#define IOCTL_USB_GET_NODE_INFORMATION   CTL_CODE ( FILE_DEVICE_USB,  \
                 USB_IOCTL_INDEX + 3, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_USB_GET_ROOT_HUB_NAME  CTL_CODE ( FILE_DEVICE_USB,  \
                 USB_IOCTL_INDEX + 3, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_USB_GET_NODE_CONNECTION_INFORMATION
    CTL_CODE ( FILE_DEVICE_USB, USB_IOCTL_INDEX + 4, METHOD_BUFFERED,  \
                                                     FILE_ANY_ACCESS)
#define IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION
    CTL_CODE ( FILE_DEVICE_USB, USB_IOCTL_INDEX + 5, METHOD_BUFFERED,  \
                                                     FILE_ANY_ACCESS)
#define IOCTL_USB_GET_NODE_CONNECTION_NAME CTL_CODE ( FILE_DEVICE_USB,  \
                USB_IOCTL_INDEX + 6, METHOD_BUFFERED, FILE_ANY_ACCESS)
#define IOCTL_USB_GET_NODE_CONNECTION_DRIVERKEY_NAME
    CTL_CODE ( FILE_DEVICE_USB, USB_IOCTL_INDEX + 9, METHOD_BUFFERED,  \
                                                     FILE_ANY_ACCESS)
#define IOCTL_GET_HCD_DRIVERKEY_NAME CTL_CODE ( FILE_DEVICE_USB,  \
                USB_IOCTL_INDEX + 10, METHOD_BUFFERED, FILE_ANY_ACCESS)
// структуры данных
#pragma pack ( 1)
```

```
// описатель дескриптора устройства
typedef struct _USB_INTERFACE_DESCRIPTOR {
   UCHAR bLength;
   UCHAR bDescriptorType;
   UCHAR bInterfaceNumber;
   UCHAR bAlternateSetting;
   UCHAR bNumEndpoints;
   UCHAR bInterfaceClass;
   UCHAR bInterfaceSubClass;
   UCHAR bInterfaceProtocol;
   UCHAR iInterface;
} USB_INTERFACE_DESCRIPTOR, *PUSB_INTERFACE_DESCRIPTOR;
// дополнительный описатель дескриптора устройства
typedef struct _USB_INTERFACE_DESCRIPTOR2 {
   UCHAR  bLength;
   UCHAR  bDescriptorType;
   UCHAR  bInterfaceNumber;
   UCHAR  bAlternateSetting;
   UCHAR  bNumEndpoints;
   UCHAR  bInterfaceClass;
   UCHAR  bInterfaceSubClass;
   UCHAR  bInterfaceProtocol;
   UCHAR  iInterface;
   USHORT wNumClasses;
} USB_INTERFACE_DESCRIPTOR2, *PUSB_INTERFACE_DESCRIPTOR2;
// описание конфигурации
typedef struct _USB_CONFIGURATION_DESCRIPTOR {
   UCHAR bLength;
   UCHAR bDescriptorType;
   USHORT wTotalLength;
   UCHAR bNumInterfaces;
   UCHAR bConfigurationValue;
   UCHAR iConfiguration;
   UCHAR bmAttributes;
   UCHAR MaxPower;
} USB_CONFIGURATION_DESCRIPTOR, *PUSB_CONFIGURATION_DESCRIPTOR;
// описатель основного дескриптора устройства
typedef struct _USB_COMMON_DESCRIPTOR {
   UCHAR bLength;
   UCHAR bDescriptorType;
} USB_COMMON_DESCRIPTOR, *PUSB_COMMON_DESCRIPTOR;
// описатель корневого устройства
typedef struct _USB_ROOT_HUB_NAME {
   ULONG ActualLength;
```

```
   WCHAR RootHubName[1];
} USB_ROOT_HUB_NAME, *PUSB_ROOT_HUB_NAME;
// параметры подключенного к хабу устройства
typedef struct _USB_NODE_CONNECTION_NAME {
   ULONG ConnectionIndex;
   ULONG ActualLength;
   WCHAR NodeName[1];
} USB_NODE_CONNECTION_NAME, *PUSB_NODE_CONNECTION_NAME;
// описание драйвера устройства
typedef struct _USB_NODE_CONNECTION_DRIVERKEY_NAME {
   ULONG ConnectionIndex;
   ULONG ActualLength;
   WCHAR DriverKeyName[1];
} USB_NODE_CONNECTION_DRIVERKEY_NAME,
 *PUSB_NODE_CONNECTION_DRIVERKEY_NAME;
// описание дескриптора
typedef struct _USB_STRING_DESCRIPTOR {
   UCHAR bLength;
   UCHAR bDescriptorType;
   WCHAR bString[1];
} USB_STRING_DESCRIPTOR, *PUSB_STRING_DESCRIPTOR;
// описание подключенного устройства
typedef struct _STRING_DESCRIPTOR_NODE
{
   struct _STRING_DESCRIPTOR_NODE *Next;
   UCHAR DescriptorIndex;
   USHORT LanguageID;
   USB_STRING_DESCRIPTOR StringDescriptor[0];
} STRING_DESCRIPTOR_NODE, *PSTRING_DESCRIPTOR_NODE;
// описание дескриптора хаба
typedef struct _USB_HUB_DESCRIPTOR {
   UCHAR bDescriptorLength;
   UCHAR bDescriptorType;
   UCHAR bNumberOfPorts;
   USHORT wHubCharacteristics;
   UCHAR bPowerOnToPowerGood;
   UCHAR bHubControlCurrent;
   UCHAR bRemoveAndPowerMask[64];
} USB_HUB_DESCRIPTOR, *PUSB_HUB_DESCRIPTOR;
// дополнительная информация о хабе
typedef struct _USB_HUB_INFORMATION {
   USB_HUB_DESCRIPTOR HubDescriptor;
   BOOLEAN HubIsBusPowered;
} USB_HUB_INFORMATION, *PUSB_HUB_INFORMATION;
```

```
typedef struct _USB_MI_PARENT_INFORMATION {
   ULONG NumberOfInterfaces;
} USB_MI_PARENT_INFORMATION, *PUSB_MI_PARENT_INFORMATION;
// параметры драйвера устройства
typedef struct _USB_HCD_DRIVERKEY_NAME {
   ULONG ActualLength;
   WCHAR DriverKeyName[1];
} USB_HCD_DRIVERKEY_NAME, *PUSB_HCD_DRIVERKEY_NAME;
// описание хаба
typedef enum _USB_HUB_NODE
{
    UsbHub,
    UsbMIParent
} USB_HUB_NODE;

typedef struct _USB_NODE_INFORMATION
{
   USB_HUB_NODE NodeType;
   union {
     USB_HUB_INFORMATION HubInformation;
     USB_MI_PARENT_INFORMATION MiParentInformation;
   } u;
} USB_NODE_INFORMATION, *PUSB_NODE_INFORMATION;
// параметры устройства
typedef struct _USB_DEVICE_DESCRIPTOR {
   UCHAR bLength;
   UCHAR bDescriptorType;
   USHORT bcdUSB;
   UCHAR bDeviceClass;
   UCHAR bDeviceSubClass;
   UCHAR bDeviceProtocol;
   UCHAR bMaxPacketSize0;
   USHORT idVendor;
   USHORT idProduct;
   USHORT bcdDevice;
   UCHAR iManufacturer;
   UCHAR iProduct;
   UCHAR iSerialNumber;
   UCHAR bNumConfigurations;
} USB_DEVICE_DESCRIPTOR, *PUSB_DEVICE_DESCRIPTOR;
// состояние соединения
typedef enum _USB_CONNECTION_STATUS {
   NoDeviceConnected,
   DeviceConnected,
```

```
   DeviceFailedEnumeration,
   DeviceGeneralFailure,
   DeviceCausedOvercurrent,
   DeviceNotEnoughPower,
   DeviceNotEnoughBandwidth
} USB_CONNECTION_STATUS, *PUSB_CONNECTION_STATUS;
// дополнительная информация о подключенном устройстве
typedef struct _USB_ENDPOINT_DESCRIPTOR {
   UCHAR bLength;
   UCHAR bDescriptorType;
   UCHAR bEndpointAddress;
   UCHAR bmAttributes;
   USHORT wMaxPacketSize;
   UCHAR bInterval;
} USB_ENDPOINT_DESCRIPTOR, *PUSB_ENDPOINT_DESCRIPTOR;
typedef struct _USB_PIPE_INFO {
   USB_ENDPOINT_DESCRIPTOR EndpointDescriptor;
   ULONG ScheduleOffset;
} USB_PIPE_INFO, *PUSB_PIPE_INFO;
typedef struct _USB_NODE_CONNECTION_INFORMATION {
   ULONG ConnectionIndex;
   USB_DEVICE_DESCRIPTOR DeviceDescriptor;
   UCHAR CurrentConfigurationValue;
   BOOLEAN LowSpeed;
   BOOLEAN DeviceIsHub;
   USHORT DeviceAddress;
   ULONG NumberOfOpenPipes;
   USB_CONNECTION_STATUS ConnectionStatus;
   USB_PIPE_INFO PipeList[0];
} USB_NODE_CONNECTION_INFORMATION, *PUSB_NODE_CONNECTION_INFORMATION;
// текущее состояние устройства
typedef struct _USB_DESCRIPTOR_REQUEST {
   ULONG ConnectionIndex;
   struct {
     UCHAR bmRequest;
     UCHAR bRequest;
     USHORT wValue;
     USHORT wIndex;
     USHORT wLength;
    } SetupPacket;
   UCHAR Data[0];
} USB_DESCRIPTOR_REQUEST, *PUSB_DESCRIPTOR_REQUEST;
```

```
// общие параметры устройства
typedef struct
{
   PCHAR HubName;
   PUSB_NODE_INFORMATION HubInfo;
   PUSB_NODE_CONNECTION_INFORMATION ConnectionInfo;
   PUSB_DESCRIPTOR_REQUEST ConfigDesc;
   PSTRING_DESCRIPTOR_NODE StringDescs;
} USBDEVICEINFO, *PUSBDEVICEINFO;
#pragma pack ()
```

Теперь, когда все основные структуры данных определены, создадим новый класс для работы с устройствами, размещенными на шине USB. Чтобы продемонстрировать основные принципы программирования данного типа оборудования, рассмотрим два базовых метода, совмещенных в одном классе. Назовем наш класс USB и напишем его реализацию так, как показано в листингах 23.9 и 23.10.

Листинг 23.9. Файл USB.h

```
// подключаем необходимые файлы
#include <objbase.h>
#include <initguid.h>
#include <setupapi.h>
#include "usbdefs.h"
// произвольные константы
#define USB_MAX_HUB 10 // максимальное число хабов
#define USB_MAX_DEVICES 10 // максимальное число устройств
#define USB_MAX_DRIVER 10 // максимальное число драйверов
// объявление класса
class USB
{
public:
   USB (); // конструктор
   ~USB () { } // пустой деструктор
// функции открытого интерфейса
   // получение общего числа найденных устройств
   unsigned int GetCountDevices ();
   // общее число подключенных устройств
   unsigned long GetConnectedDevices ();
   // получение имени устройства
   void GetDriverName ( unsigned int num, char* driver);
   // получение имени корневого устройства
   void GetRootHubName ( unsigned int num, char* root_hub);
```

```
   // получение идентификатора производителя
   unsigned short GetVendorID ( unsigned int num);
   // получение идентификатора устройства
   unsigned short GetProductID ( unsigned int num);
   // получение номера версии устройства
   unsigned short GetVersionNumber ( unsigned int num);
   // получение строкового описания производителя устройства
   void GetVendorName ( unsigned int num, char* vendor);
   // получение строкового описания устройства
   void GetProductName ( unsigned int num, char* product);
   // получение серийного номера устройства
   void GetSerialNumber ( unsigned int num, char* serial);
   bool OpenDevice ( unsigned int num); // открытие устройства
   void CloseDevice (); // закрытие текущего устройства
   // получение данных о возможностях устройства
   bool GetDeviceCaps ();
   // установка нового параметра для устройства
   bool SetFeature ( void* data, unsigned long data_len);
   // получение указанного параметра устройства
   bool GetFeature ( void* data);
   // чтение данных из устройства
   bool ReadData ( char* buffer);
   // передача данных устройству
   bool WriteData ( char* buffer, unsigned long buffer_len);
// расширенные функции для демонстрации второго метода доступа
   // получение имени устройства
   void GetDeviceName_Ex ( unsigned int num, char* name);
   // получение идентификатора производителя
   unsigned short GetVendor_Ex ( unsigned int num);
   // получение идентификатора устройства
   unsigned short GetProduct_Ex ( unsigned int num);
   // получение класса устройства
   unsigned char GetClass_Ex ( unsigned int num);
   // получение адреса устройства
   unsigned short GetAddress_Ex ( unsigned int num);
private:
// описатель устройства для первого метода
typedef struct
{
   unsigned short Vendor_ID;
   unsigned short Product_ID;
   unsigned short VersionNumber;
   char DevicePath[ANYSIZE_ARRAY];
```

```
   char VendorName[MAX_PATH];
   char ProductName[MAX_PATH];
   char SerialNumber[50];
} USBDEVPARAMS, *PUSBDEVPARAMS;
// параметры устройства
typedef struct
{
   char Device[MAX_PATH];
   unsigned short Vendor;
   unsigned short Product;
   unsigned char SerialNumber;
   unsigned char DeviceClass;
   unsigned char DeviceSubClass;
   unsigned char MaxPacketSize;
   unsigned short Address;
   unsigned long CountOpenChannel;
   USB_CONNECTION_STATUS Status;
} USBPORT, *PUSBPORT;
// описатель устройства
typedef struct
{
   char HubName[MAX_PATH];
   unsigned short Address;
   bool bIsHub;
   bool bLowSpeed;
   unsigned long CountOpenChannel;
   USB_DEVICE_DESCRIPTOR d;
   USB_NODE_INFORMATION node;
   USB_CONNECTION_STATUS Status;
} USBDEVICE, *PUSBDEVICE;
// описатель устройства для второго метода
typedef struct
{
   char DriverName[MAX_PATH];
   char RootHubName[MAX_PATH];
   USBDEVICE dev[USB_MAX_HUB];
} USBDRIVERPARAMS, *PUSBDRIVERPARAMS, *LPUSBDRIVERPARAMS;
// описатель возможностей устройства
typedef struct
{
   unsigned short InputByteLength;
   unsigned short OutputByteLength;
   unsigned short FeatureByteLength;
} USBDATASIZE, *PUSBDATASIZE;
```

```
// объявляем рабочие структуры
USBDEVPARAMS usb[USB_MAX_DEVICES];
USBDRIVERPARAMS drv[USB_MAX_HUB];
USBPORT port[USB_MAX_DEVICES];
USBDATASIZE usb_size;
// объявляем переменные класса
HANDLE m_hDevice; // дескриптор устройства
bool m_bIsGetParams; // данные для устройства по первому методу
bool m_bIsGetParamsEx; // данные для устройства по второму методу
bool m_ConfigDesc; // данные о конфигурации устройства
bool m_bIsCaps; // данные о возможностях устройства
unsigned int m_uCountDevices; // количество найденных устройств
unsigned int m_iCountHub; // количество хабов
int m_iCurrentOpenDevice; // номер текущего открытого устройства
unsigned long m_lCountDevicesConnected; // число подключенных устройств
// вспомогательные функции класса
// определение параметров устройства по первому методу
void _getDevicesParams ();
// определение параметров устройства по второму методу
void _getDevicesParamsEx ();
// получение имени устройства
char* _getKeyName ( HANDLE  hDevice);
// получение имени драйвера
char* _getDrvKeyName ( HANDLE  hHub, unsigned long ConnectionIndex);
// конвертирование строки
char* _wStr_to_mStr ( PWCHAR wStr);
// получение имени корневого хаба
char* _getRootHubName ( HANDLE HostController);
// перечисление устройств, подключенных к хабу
void  _enumHubDevices ( char* hub_name,
PUSB_NODE_CONNECTION_INFORMATION connect_info,
PUSB_DESCRIPTOR_REQUEST cfg, PSTRING_DESCRIPTOR_NODE node, char* device,
unsigned int index);
// перечисление портов
void _enumHubPorts ( HANDLE hHub, unsigned long NumberPorts);
// получение описателя конфигурации
PUSB_DESCRIPTOR_REQUEST _getConfigDescriptor ( HANDLE hHub,
             unsigned long ConnectionIndex, unsigned char Index);
// проверка типа описателя
bool _IsDescriptors ( PUSB_DEVICE_DESCRIPTOR Device,
                        PUSB_CONFIGURATION_DESCRIPTOR Config);
// получение всех описателей
PSTRING_DESCRIPTOR_NODE _getAllStringDescriptors ( HANDLE hHub,
```

```
 unsigned long ConnectionIndex, PUSB_DEVICE_DESCRIPTOR Device,
 PUSB_CONFIGURATION_DESCRIPTOR Config);
// получение имени внешнего хаба
char* _getExternalHubName ( HANDLE hHub,
                            unsigned long ConnectionIndex);
// получение описателя
PSTRING_DESCRIPTOR_NODE _getStringDescriptor ( HANDLE hHub,
 unsigned long ConnectionIndex, unsigned char Index,
 unsigned short LanguageID);
// получение описателя
PSTRING_DESCRIPTOR_NODE _getStringDescriptors ( HANDLE  hHub,
 unsigned long ConnectionIndex, unsigned char Index,
 unsigned long CountLangIDs, unsigned short* LangIDs,
 PSTRING_DESCRIPTOR_NODE Node);
}; // завершение класса
```

Листинг 23.10. Файл USB.cpp

```
#include "stdafx.h"
#include "USB.h"
// реализация класса
USB :: USB ()
{
   // обнуляем переменные и структуры класса
   m_bIsGetParams = false;
   m_bIsGetParamsEx = false;
   m_ConfigDesc = false;
   m_bIsCaps = false;
   m_uCountDevices = 0;
   m_iCountHub = 0;
   m_lCountDevicesConnected = 0;
   m_hDevice = NULL;
   m_iCurrentOpenDevice = -1;
   ZeroMemory ( &usb, sizeof ( USBDEVPARAMS));
   ZeroMemory ( &drv, sizeof ( USBDRIVERPARAMS));
   ZeroMemory ( &port, sizeof ( USBPORT));
}
void USB :: _getDevicesParams ()
{
   SP_DEVICE_INTERFACE_DATA dev;
   PSP_DEVICE_INTERFACE_DETAIL_DATA detail = NULL;
   HIDD_ATTRIBUTES atr;
   HANDLE hDevice = NULL, hInfo = NULL;
```

```
GUID guid;
int index = 0;
long result = 0L;
DWORD size_buffer = 0, req_size = 0;
char buf[MAX_PATH];
bool last = false;
// обнуляем структуру
ZeroMemory ( &dev, sizeof ( SP_DEVICE_INTERFACE_DATA));
// пôлучаем глобальный идентификатор для всех устройств USB
HidD_GetHidGuid ( &guid);
// получаем информацию для всех устройств
hInfo = SetupDiGetClassDevs ( &guid, NULL, NULL, DIGCF_INTERFACEDEVICE
                              | DIGCF_PRESENT);
// в случае ошибки, выходим из функции
if ( hInfo == INVALID_HANDLE_VALUE) return;
// устанавливаем размер структуры перед использованием
dev.cbSize = sizeof ( SP_DEVICE_INTERFACE_DATA);
// перебираем устройства
do
{
   // получаем информацию об интерфейсе устройства
   result = SetupDiEnumDeviceInterfaces ( hInfo, 0, &guid, index,
                                          &dev);
   // если информация найдена
   if ( result != 0)
   {
      // определяем необходимый размер буфера для получения данных
      result = SetupDiGetDeviceInterfaceDetail ( hInfo, &dev, NULL, 0,
                                                 &size_buffer, NULL);
      // если ошибка, переходим к следующему устройству
      if ( !result) continue;
      // выделяем необходимый размер памяти для получения всех данных
      detail =
      // устанавливаем размер структуры
      detail->cbSize = sizeof ( SP_DEVICE_INTERFACE_DETAIL_DATA);
      // повторно вызываем функцию
      result = SetupDiGetDeviceInterfaceDetail ( hInfo, &dev, detail,
                                            size_buffer, &req_size, NULL);
      if ( !result) // если ошибка
      {
        if ( detail) free ( detail); // освобождаем память
        continue; // переходим к следующему устройству
      }
```

```
            // сохраняем путь к устройству для последующего использования
            strcpy ( usb[m_uCountDevices].DevicePath, detail->DevicePath);
            // открываем найденное устройство
            hDevice = CreateFile ( detail->DevicePath,
             GENERIC_READ|GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE,
             ( LPSECURITY_ATTRIBUTES) NULL, OPEN_EXISTING, 0, NULL);
             if ( hDevice == INVALID_HANDLE_VALUE) // если ошибка
             {
               if ( detail) free ( detail); // освобождаем память
               continue; // переходим к следующему устройству
             }
            // устанавливаем размер структуры HIDD_ATTRIBUTES
            atr.Size = sizeof ( HIDD_ATTRIBUTES);
            // получаем основные параметры устройства
            HidD_GetAttributes ( hDevice, &atr);
            // сохраняем их для последующего применения
            usb[m_uCountDevices].Vendor_ID = atr.VendorID;
            usb[m_uCountDevices].Product_ID = atr.ProductID;
            usb[m_uCountDevices].VersionNumber = atr.VersionNumber;
            // получаем имя производителя
            HidD_GetManufacturerString ( hDevice, buf, sizeof ( buf));
            strcpy ( usb[m_uCountDevices].VendorName, buf);
            strcpy ( buf, " ");
            // получаем наименование изделия
            HidD_GetProductString ( hDevice, buf, sizeof ( buf));
            strcpy ( usb[m_uCountDevices].ProductName, buf);
            strcpy ( buf, " ");
            // получаем серийный номер изделия
            HidD_GetSerialNumberString ( hDevice, buf, sizeof ( buf));
            strcpy ( usb[m_uCountDevices].SerialNumber, buf);
            strcpy ( buf, " ");
            // закрываем устройство и освобождаем память
            CloseHandle ( hDevice);
            free ( detail);
            m_uCountDevices++;
        }
    } while ( last = false);
    // освобождаем выделенную память
    SetupDiDestroyDeviceInfoList ( hInfo);
    // устанавливаем признак выполнения операции
    m_bIsGetParams = true;
}
```

```
void USB :: _getDevicesParamsEx ()
{
   HANDLE hDev = NULL;
   char* RootHubName = NULL;
   char* driver = NULL;
   char name[16];
   int  i = 0;
   // перечисляем все возможные драйверы
   for ( i = 0; i < USB_MAX_DRIVER; i++)
   {
      // форматируем корректное имя драйвера
      wsprintf ( name, "\\\\.\\HCD%d", i);
      // открываем драйвер
      hDev = CreateFile ( name, GENERIC_WRITE, FILE_SHARE_WRITE,
                          NULL, OPEN_EXISTING, 0, NULL);
      // если ошибок нет
      if ( hDev != INVALID_HANDLE_VALUE)
      {
        // получаем имя драйвера
        driver = _getKeyName ( hDev);
        // сохраняем его для последующего использования
        strcpy ( drv[i].DriverName, driver);
        // если необходимо, освобождаем память
        if ( driver) GlobalFree ( driver);
        // получаем имя корневого устройства
        RootHubName = _getRootHubName ( hDev);
        // если ошибок нет
        if (RootHubName != NULL)
              _enumHubDevices ( RootHubName, NULL, NULL, NULL, "RootHub",
                                i); // получаем все устройства на хабе
        // сохраняем имя корневого хаба для последующего использования
        strcpy ( drv[i].RootHubName,  RootHubName);
        CloseHandle ( hDev); // закрываем драйвер
      }
   }
   // устанавливаем признак выполнения операции
   m_bIsGetParamsEx = true;
}
void  USB :: _enumHubDevices ( char* hub_name,
 PUSB_NODE_CONNECTION_INFORMATION connect_info,
 PUSB_DESCRIPTOR_REQUEST cfg, PSTRING_DESCRIPTOR_NODE node, char* device,
 unsigned int index)
{
   PUSBDEVICEINFO info = NULL;
```

```
 HANDLE hHub = INVALID_HANDLE_VALUE;
 char* device_name = NULL;
 char Name[512];
 bool result;
 unsigned long ulBytes = 0;
 // выделяем память для информационной структуры и обнуляем ее
 info = ( PUSBDEVICEINFO) GlobalAlloc ( GPTR,
                                        sizeof ( USBDEVICEINFO));
 // если ошибка, переходим к обработчику ошибок
 if ( info == NULL) goto Error;
 // передаем данные в структуру
 info->HubName = hub_name;
 info->ConnectionInfo = connect_info;
 info->ConfigDesc = cfg;
 info->StringDescs = node;
 // выделяем память для структуры USB_NODE_INFORMATION
info->HubInfo = ( PUSB_NODE_INFORMATION) GlobalAlloc( GPTR,
                  sizeof ( USB_NODE_INFORMATION));
 // если ошибка, переходим к обработчику ошибок
 if ( info->HubInfo == NULL) goto Error;
 // выделяем память для имени хаба
 device_name = ( PCHAR) GlobalAlloc ( GPTR, strlen ( hub_name) +
                                      sizeof ( "\\\\.\\"));
 // если ошибка, переходим к обработчику ошибок
 if ( device_name == NULL) goto Error;
 // создаем полное имя хаба
 strcpy ( device_name, "\\\\.\\");
 strcpy ( device_name + sizeof ( "\\\\.\\") - 1, info->HubName);
 // открываем устройство
 hHub = CreateFile ( device_name, GENERIC_WRITE, FILE_SHARE_WRITE,
                     NULL, OPEN_EXISTING, 0, NULL);
 // освобождаем память
 GlobalFree ( device_name);
 // если ошибка, переходим к обработчику ошибок
 if ( hHub == INVALID_HANDLE_VALUE) goto Error;
 // делаем запрос на устройство
 result = DeviceIoControl ( hHub, IOCTL_USB_GET_NODE_INFORMATION,
  info->HubInfo, sizeof ( USB_NODE_INFORMATION), info->HubInfo,
  sizeof ( USB_NODE_INFORMATION), &ulBytes, NULL);
 // если ошибка, переходим к обработчику ошибок
  if ( !result) goto Error;
 // форматируем полученные данные
 if ( connect_info)
```

```
    {
      wsprintf ( Name, "[Port%d] ", connect_info->ConnectionIndex);
      strcat ( Name, " :  ");
    }
    else
      Name[0] = 0;
    if ( device)
     strcat ( Name, device);
    else
     strcat ( Name, info->HubName);
    // сохраняем имя хаба для последующего использования
    strcpy ( drv[index].dev[index].HubName, Name);
    // сохраняем параметры устройства
    if ( connect_info)
    {
      drv[index].dev[index].Status =
         info->ConnectionInfo->ConnectionStatus;
      drv[index].dev[index].Address = info->ConnectionInfo->DeviceAddress;
      drv[index].dev[index].bIsHub = info->ConnectionInfo->DeviceIsHub;
      drv[index].dev[index].bLowSpeed = info->ConnectionInfo->LowSpeed;
      drv[index].dev[index].CountOpenChannel =
         info->ConnectionInfo->NumberOfOpenPipes;
      memcpy ( &drv[index].dev[index].d,
        &info->ConnectionInfo->DeviceDescriptor, sizeof ( USBDEVICE));
    }
    // перечисляем порты для хаба
    _enumHubPorts ( hHub,
        info->HubInfo->u.HubInformation.HubDescriptor.bNumberOfPorts);
    // закрываем устройство
    CloseHandle ( hHub);
    return; // выходим из функции
// обработка ошибок
Error:
    if ( hHub != INVALID_HANDLE_VALUE)
    {
      CloseHandle ( hHub);
      hHub = NULL;
    }
    if ( info != NULL)
    {
        if ( info->HubName != NULL)
        {
          GlobalFree ( info->HubName);
```

```
        info->HubName = NULL;
      }
      if ( info->HubInfo != NULL)
      {
        GlobalFree ( info->HubInfo);
        info->HubInfo;
      }
      GlobalFree ( info);
      info = NULL;
  }
  if ( connect_info) GlobalFree ( connect_info);
  if ( cfg)  GlobalFree ( cfg);
  if ( node != NULL)
  {
    PSTRING_DESCRIPTOR_NODE Next;
   do {
       Next = node->Next;
       GlobalFree ( node);
       node = Next;
      } while ( пode != NULL);
  }
}
void USB :: _enumHubPorts ( HANDLE  hHub, unsigned long NumberPorts)
{

  PUSB_NODE_CONNECTION_INFORMATION connect_info = NULL;
  PUSB_DESCRIPTOR_REQUEST config = NULL;
  PSTRING_DESCRIPTOR_NODE str = NULL;
  PUSBDEVICEINFO info = NULL;
  char* driver = NULL;
  char* device = NULL;
  char  Name[512];
  unsigned long index;
  bool result;
  // определяем все доступные порты
  for ( index = 1; index <= NumberPorts; index++)
  {
    unsigned long ulBytes;
    // вычисляем размер данных
    ulBytes = sizeof ( USB_NODE_CONNECTION_INFORMATION) +
              sizeof ( USB_PIPE_INFO) * 30;
    // выделяем память для информационной структуры
    connect_info =
     ( PUSB_NODE_CONNECTION_INFORMATION) GlobalAlloc ( GPTR, ulBytes);
```

```
// если ошибка, выходим из цикла
if ( connect_info == NULL) break;
// определяем номер порта
connect_info->ConnectionIndex = index;
// делаем запрос для данного порта
result = DeviceIoControl ( hHub,
  IOCTL_USB_GET_NODE_CONNECTION_INFORMATION, connect_info, ulBytes,
  connect_info, ulBytes, &ulBytes, NULL);
// если ошибка
if ( !result)
{
  GlobalFree ( connect_info); // освобождаем память
  continue; // переходим к следующему порту
}
// вычисляем число подключенных устройств
if ( connect_info->ConnectionStatus == DeviceConnected)
     m_lCountDevicesConnected++;
// вычисляем число хабов
if ( connect_info->DeviceIsHub)
     m_iCountHub++;
// если устройство подключено, получаем описание
device = NULL;
if ( connect_info->ConnectionStatus != NoDeviceConnected)
{
  driver = _getDrvKeyName ( hHub,index);
  strcpy ( drv[index].dev->HubName,  driver);
  if ( driver) GlobalFree ( driver); // освобождаем память
}
// если устройство подключено, определяем его конфигурацию
if ( m_ConfigDesc &&
   connect_info->ConnectionStatus == DeviceConnected)
{
   m_ConfigDesc = _getConfigDescriptor ( hHub, index, 0);
}
else
  m_ConfigDesc = NULL;
// получаем описатели устройства
if ( m_ConfigDesc != NULL &&
   _IsDescriptors ( &connect_info->DeviceDescriptor,
        ( PUSB_CONFIGURATION_DESCRIPTOR) ( config + 1)))
{
 str = _getAllStringDescriptors ( hHub, index,
                                  &connect_info->DeviceDescriptor,
        ( USB_CONFIGURATION_DESCRIPTOR) ( config + 1));
}
```

```
else
 str = NULL;
// если устройство связано с внешним хабом, получаем данные о нем
if ( connect_info->DeviceIsHub)
{
   char* pHubName = NULL;
   pHubName = _getExternalHubName ( hHub, index);
   if ( pHubName != NULL) // если хаб существует
   {
     // перечисляем подключенные к нему устройства
     _enumHubDevices ( pHubName, connect_info, config, str, device,
                        index);
     continue; // переходим к следующему порту
   }
}
// выделяем память для информационной структуры
info = ( PUSBDEVICEINFO) GlobalAlloc ( GPTR,
                                        sizeof ( USBDEVICEINFO));
// если ошибка, освобождаем ресурсы
if ( info == NULL)
{
   if ( config != NULL)
     GlobalFree (config);
   GlobalFree ( connect_info);
   break;
}
// заполняем поля структуры полученными данными
info->ConnectionInfo = connect_info;
info->ConfigDesc = config;
info->StringDescs = str;
// форматируем имя порта
wsprintf ( Name, "[Port%d] ", index);
if ( device)
{
  strcat ( Name, " :  ");
  strcat( Name, device);
}
// сохраняем полученные данные для последующего использования
strcpy ( port[index].Device, Name);
port[index].Address = info->ConnectionInfo->DeviceAddress;
port[index].CountOpenChannel =
                  info->ConnectionInfo->NumberOfOpenPipes;
port[index].DeviceClass =
      info->ConnectionInfo->DeviceDescriptor.bDeviceClass;
```

```
    port[index].DeviceSubClass =
        info->ConnectionInfo->DeviceDescriptor.bDeviceSubClass;
    port[index].MaxPacketSize =
         info->ConnectionInfo->DeviceDescriptor.bMaxPacketSize0;
    port[index].Product =
              info->ConnectionInfo->DeviceDescriptor.idProduct;
    port[index].SerialNumber =
          info->ConnectionInfo->DeviceDescriptor.iSerialNumber;
    port[index].Status = info->ConnectionInfo->ConnectionStatus;
    port[index].Vendor =
               info->ConnectionInfo->DeviceDescriptor.idVendor;
    // освобождаем память
    GlobalFree ( info);
  }
}
char* USB :: _getExternalHubName ( HANDLE  hHub,
                                    unsigned long ConnectionIndex)
{
   USB_NODE_CONNECTION_NAME HubName;
   PUSB_NODE_CONNECTION_NAME HubName2 = NULL;
   char* HubName3 = NULL;
   bool result;
   unsigned long ulBytes = 0;
   // определяем размер данных для хранения имени хаба
   HubName.ConnectionIndex = ConnectionIndex;
   result = DeviceIoControl ( hHub,
    IOCTL_USB_GET_NODE_CONNECTION_NAME, &HubName, sizeof ( HubName),
    &HubName, sizeof ( HubName), &ulBytes, NULL);
   // если ошибка, переходим к обработчику ошибок
   if ( !result) goto Error;
   // получаем размер в байтах для хранения имени хаба
   ulBytes = HubName.ActualLength;
   // если ошибка, переходим к обработчику ошибок
   if ( ulBytes <= sizeof ( HubName)) goto Error;
   // выделяем память
   HubName2 = ( PUSB_NODE_CONNECTION_NAME) GlobalAlloc ( GPTR,
                                                           ulBytes);
   // если ошибка, переходим к обработчику ошибок
   if ( HubName2 == NULL) goto Error;
   HubName2->ConnectionIndex = ConnectionIndex;
   // получаем имя внешнего хаба
   result = DeviceIoControl ( hHub,
     IOCTL_USB_GET_NODE_CONNECTION_NAME, HubName2, ulBytes, HubName2,
     ulBytes, &ulBytes, NULL);
```

```
    // если ошибка, переходим к обработчику ошибок
    if ( !result) goto Error;
    // конвертируем имя внешнего хаба
    HubName3 = _wStr_to_mStr ( HubName2->NodeName);
    // освобождаем память
    GlobalFree ( HubName2);
    // возвращаем имя внешнего хаба
    return HubName3;
// обработчик ошибок
Error:
    if ( HubName2 != NULL)
    {
      GlobalFree ( HubName2);
      HubName2 = NULL;
    }
    return NULL; // завершаем функцию с ошибкой
}
PSTRING_DESCRIPTOR_NODE USB :: _getStringDescriptor ( HANDLE  hHub,
 unsigned long ConnectionIndex, unsigned char Index,
 unsigned short LanguageID)
{
    PUSB_DESCRIPTOR_REQUEST Req = NULL;
    PUSB_STRING_DESCRIPTOR  str = NULL;
    PSTRING_DESCRIPTOR_NODE Node = NULL;
    bool result;
    unsigned long ulBytes = 0, ulBytesReturned = 0;
    unsigned char ReqBuf[ sizeof ( USB_DESCRIPTOR_REQUEST) +
                              MAXIMUM_USB_STRING_LENGTH];
    ulBytes = sizeof ( ReqBuf);
    // инициализируем переменные
    Req = ( PUSB_DESCRIPTOR_REQUEST) ReqBuf;
    str = ( PUSB_STRING_DESCRIPTOR) ( Req + 1);
    // обнуляем структуру
    memset ( Req, 0, ulBytes);
    // указываем номер порта
    Req->ConnectionIndex = ConnectionIndex;
    // инициализируем поля структуры
    Req->SetupPacket.wValue = ( USB_STRING_DESCRIPTOR_TYPE << 8) | Index;
    Req->SetupPacket.wIndex = LanguageID;
    // вычисляем размер данных
     Req->SetupPacket.wLength = ( USHORT) ( ulBytes -
                                   sizeof ( USB_DESCRIPTOR_REQUEST));
```

```
    // делаем запрос для получения дескриптора
    result = DeviceIoControl ( hHub,
      IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, Req, ulBytes, Req,
      ulBytes, &ulBytesReturned, NULL);
    // если ошибка, выходим из функции с ошибкой
    if ( !result) return NULL;
    // если некорректный размер полученных данных
    // выходим из функции с ошибкой
    if ( ulBytesReturned < 2) return NULL;
    // если полученный дескриптор имеет неправильный тип
    if ( str->bDescriptorType != USB_STRING_DESCRIPTOR_TYPE)
         return NULL; // выходим из функции с ошибкой
    // если не совпал размер дескриптора
    if (str->bLength != ulBytesReturned - sizeof(USB_DESCRIPTOR_REQUEST))
         return NULL; выходим из функции с ошибкой
    // если есть остаток
     if ( str->bLength % 2 != 0)
         return NULL; // выходим из функции с ошибкой
    // выделяем память для сохранения дескриптора
    Node = ( PSTRING_DESCRIPTOR_NODE) GlobalAlloc ( GPTR,
             sizeof ( STRING_DESCRIPTOR_NODE) +  str->bLength);
    if ( Node == NULL) // выходим из функции с ошибкой
    // заполняем выделенную память данными
     Node->DescriptorIndex = Index;
     Node->LanguageID = LanguageID;
     memcpy ( Node->StringDescriptor, str, str->bLength);
    return Node; // возвращаем требуемые данные о дескрипторе
}
PSTRING_DESCRIPTOR_NODE USB :: _getAllStringDescriptors (
 HANDLE  hHub, unsigned long ConnectionIndex,
 PUSB_DEVICE_DESCRIPTOR Device, PUSB_CONFIGURATION_DESCRIPTOR Config)
{
    PSTRING_DESCRIPTOR_NODE languages = NULL;
    PSTRING_DESCRIPTOR_NODE Node = NULL;
    PUSB_COMMON_DESCRIPTOR  common = NULL;
    unsigned long countLanguageIDs = 0;
    unsigned short* langIDs = NULL;
    unsigned char* last = NULL;
    // получаем идентификаторы языков
    languages = _getStringDescriptor ( hHub, ConnectionIndex, 0, 0);
    // если ошибка, выходим из функции
    if ( languages == NULL) return NULL;
    countLanguageIDs = ( languages->StringDescriptor->bLength - 2) / 2;
```

```
langIDs = &languages->StringDescriptor->bString[0];
Node = languages;
// получаем описатели устройства
if ( Device->iManufacturer)
{
   Node = _getStringDescriptors ( hHub, ConnectionIndex,
              Device->iManufacturer, countLanguageIDs, langIDs, Node);
}
if ( Device->iProduct)
{
    Node = _getStringDescriptors ( hHub, ConnectionIndex,
              Device->iProduct, countLanguageIDs, langIDs, Node);
}
if ( Device->iSerialNumber)
{
     Node = _getStringDescriptors ( hHub, ConnectionIndex,
              Device->iSerialNumber, countLanguageIDs, langIDs, Node);
}
// определяем конфигурацию и описатель интерфейса
last = ( PUCHAR) Config + Config->wTotalLength;
common= ( PUSB_COMMON_DESCRIPTOR) Config;
while ( ( PUCHAR) common + sizeof ( USB_COMMON_DESCRIPTOR) < last &&
        ( PUCHAR) common + common->bLength <= last)
{
  switch (common->bDescriptorType)
   {
  case USB_CONFIGURATION_DESCRIPTOR_TYPE: // описатель конфигурации
    if ( common->bLength != sizeof ( USB_CONFIGURATION_DESCRIPTOR))
         break;
   if ( ( ( PUSB_CONFIGURATION_DESCRIPTOR) common)->iConfiguration)
   {
      // получаем описатели
      Node = _getStringDescriptors ( hHub, ConnectionIndex,
       ( ( PUSB_CONFIGURATION_DESCRIPTOR) common)->iConfiguration,
       countLanguageIDs, langIDs, Node);
   }
   // увеличиваем общий размер прочитанных данных
   common += common->bLength;
   continue; // переходим к следующему описателю
  case USB_INTERFACE_DESCRIPTOR_TYPE: // описатель интерфейса
    if ( common->bLength != sizeof ( USB_INTERFACE_DESCRIPTOR) &&
      common->bLength != sizeof ( USB_INTERFACE_DESCRIPTOR2))
          break;
```

```
        if ( ( ( PUSB_INTERFACE_DESCRIPTOR) common)->iInterface)
        {
            // получаем описатели
            Node = _getStringDescriptors ( hHub, ConnectionIndex,
             ( ( PUSB_INTERFACE_DESCRIPTOR) common)->iInterface,
             countLanguageIDs, langIDs, Node);
         }
         // увеличиваем общий размер прочитанных данных
         common += common->bLength;
         continue; // переходим к следующему описателю
        // если описатель не имеет нужного типа
        default:
         // просто увеличиваем общий размер прочитанных данных
         common += common->bLength;
         continue; // переходим к следующему описателю
         }
         break;
     }
    // возвращаем описатель языка
    return languages;
}

PSTRING_DESCRIPTOR_NODE USB :: _getStringDescriptors ( HANDLE  hHub,
 unsigned long ConnectionIndex, unsigned char Index,
 unsigned long CountLangIDs, unsigned short* LangIDs,
 PSTRING_DESCRIPTOR_NODE Node)
{
    unsigned long i;
    // вычисляем число описателей
    for ( i = 0; i < CountLangIDs; i++)
    {
      // получаем описатель
      Node->Next = _getStringDescriptor ( hHub, ConnectionIndex, Index,
                                             *LangIDs);

      if ( Node->Next) // переходим к следующему
      {
         Node = Node->Next;
      }
      LangIDs++; // увеличиваем счетчик
    }
    return Node;
}
```

```
bool USB :: _IsDescriptors ( PUSB_DEVICE_DESCRIPTOR Device,
                             PUSB_CONFIGURATION_DESCRIPTOR Config)
{
   unsigned char* last = NULL;
   PUSB_COMMON_DESCRIPTOR  common = NULL;
   // проверяем корректность описателя
   if ( Device->iManufacturer || Device->iProduct ||
        Device->iSerialNumber)
     return true;
   // проверяем описатель конфигурации и интерфейса
   last = ( PUCHAR) Config + Config->wTotalLength;
   common = ( PUSB_COMMON_DESCRIPTOR) Config;
   while ( ( PUCHAR) common + sizeof ( USB_COMMON_DESCRIPTOR) < last &&
         ( PUCHAR) common + common->bLength <= last)
   {
    switch ( common->bDescriptorType) // проверяем тип описателя
    (
     case USB_CONFIGURATION_DESCRIPTOR_TYPE: // описатель конфигурации
      if ( common->bLength != sizeof ( USB_CONFIGURATION_DESCRIPTOR))
           break;
      // выполняем проверку
      if ( ( ( PUSB_CONFIGURATION_DESCRIPTOR) common)->iConfiguration)
           return true;
      // увеличиваем общий размер прочитанных данных
      common += common->bLength;
      continue; // переходим к следующему описателю
     case USB_INTERFACE_DESCRIPTOR_TYPE: // описатель интерфейса
      if ( common->bLength != sizeof ( USB_INTERFACE_DESCRIPTOR) &&
           common->bLength != sizeof ( USB_INTERFACE_DESCRIPTOR2))
           break;
      // выполняем проверку
      if ( ( ( PUSB_INTERFACE_DESCRIPTOR) common)->iInterface)
           return true;
      // увеличиваем общий размер прочитанных данных
      common += common->bLength;
      continue; // переходим к следующему описателю
     // если описатель не имеет нужного типа
     default:
      // просто увеличиваем общий размер прочитанных данных
      common += common->bLength;
      continue; // переходим к следующему описателю
    }
    break;
}
```

```
return false;
}
PUSB_DESCRIPTOR_REQUEST USB :: _getConfigDescriptor ( HANDLE  hHub,
                  unsigned long ConnectionIndex, unsigned char Index)
{
   PUSB_DESCRIPTOR_REQUEST Req = NULL;
   PUSB_CONFIGURATION_DESCRIPTOR config = NULL;
   bool result;
   unsigned long ulBytes = 0, ulBytesReturned = 0;
   // вычисляем размер буфера
   unsigned char ReqBuf[sizeof ( USB_DESCRIPTOR_REQUEST) +
                         sizeof ( USB_CONFIGURATION_DESCRIPTOR)];
   // определяем параметры для запроса описателя конфигурации
   ulBytes = sizeof ( ReqBuf);
   Req = ( PUSB_DESCRIPTOR_REQUEST) ReqBuf;
   config = ( PUSB_CONFIGURATION_DESCRIPTOR) ( Req + 1);
   // обнуляем структуру
   memset ( Req, 0, ulBytes);
   // указываем номер порта
   Req->ConnectionIndex = ConnectionIndex;
   // и значение по умолчанию
   Req->SetupPacket.wValue = ( USB_CONFIGURATION_DESCRIPTOR_TYPE << 8)
                              | Index;
   // размер данных
   Req->SetupPacket.wLength = ( unsigned short) ( ulBytes -
                                 sizeof ( USB_DESCRIPTOR_REQUEST));
   // получаем дескриптор конфигурации
   result = DeviceIoControl ( hHub,
     IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, Req, ulBytes, Req,
     ulBytes, &ulBytesReturned, NULL);
   // если ошибка, выходим из функции
   if ( !result) return NULL;
   // если получено не ожидаемое количество данных, выходим из функции
   if ( ulBytes != ulBytesReturned) return NULL;
   // если не совпадает размер описателя, выходим из функции
   if (config->wTotalLength < sizeof(USB_CONFIGURATION_DESCRIPTOR))
       return NULL;
   // определяем параметры для получения полного описателя конфигурации
   ulBytes = sizeof ( USB_DESCRIPTOR_REQUEST) + config->wTotalLength;
   // выделяем память
   Req  = ( PUSB_DESCRIPTOR_REQUEST) GlobalAlloc ( GPTR, ulBytes);
   // если ошибка, выходим из функции
   if ( Req == NULL) return NULL;
```

```
   // передаем указатель
   config = ( PUSB_CONFIGURATION_DESCRIPTOR) ( Req + 1);
   // указываем номер порта
   Req->ConnectionIndex = ConnectionIndex;
   // инициализируем поля структуры
   Req->SetupPacket.wValue = ( USB_CONFIGURATION_DESCRIPTOR_TYPE << 8)
                             | Index;
   // вычисляем размер данных
   Req->SetupPacket.wLength = ( unsigned short) ( ulBytes -
                                sizeof ( USB_DESCRIPTOR_REQUEST));
   // получаем описатель
   result = DeviceIoControl ( hHub,
       IOCTL_USB_GET_DESCRIPTOR_FROM_NODE_CONNECTION, Req, ulBytes, Req,
       ulBytes, &ulBytesReturned, NULL);
   // если ошибка, освобождаем память и выходим из функции
   if ( !result)
   {
     GlobalFree ( Req);
     return NULL;
   }
   // если получено не ожидаемое количество данных, выходим из функции
   if ( ulBytes != ulBytesReturned)
   {
     GlobalFree ( Req);
     return NULL;
   }
   // если не совпадает размер описателя, выходим из функции
   if (config->wTotalLength != (ulBytes -
       sizeof ( USB_DESCRIPTOR_REQUEST)))
   {
     GlobalFree ( Req);
     return NULL;
   }
   return Req; // возвращаем описатель конфигурации
}
char* USB :: _getDrvKeyName ( HANDLE  hHub,
                              unsigned long ConnectionIndex)
{
   bool result;
   unsigned long ulBytes = 0;
   USB_NODE_CONNECTION_DRIVERKEY_NAME  driverName;
   PUSB_NODE_CONNECTION_DRIVERKEY_NAME driverName2 = NULL;
   char* driverName3 = NULL;
```

```
    // определяем размер данных для имени драйвера
    driverName.ConnectionIndex = ConnectionIndex; // номер порта
    // делаем запрос
    result = DeviceIoControl ( hHub,
       IOCTL_USB_GET_NODE_CONNECTION_DRIVERKEY_NAME, &driverName,
       sizeof ( driverName), &driverName, sizeof ( driverName),
       &ulBytes, NULL);
    // если ошибка, передаем управление обработчику ошибок
    if ( !result) goto Error;
    // получаем необходимый размер буфера для данных
    ulBytes = driverName.ActualLength;
    // если размер меньше требуемого, переходим к обработчику ошибок
    if ( ulBytes <= sizeof ( driverName)) goto Error;
    // выделяем память
    driverName2 =
    ( PUSB_NODE_CONNECTION_DRIVERKEY_NAME) GlobalAlloc ( GPTR, ulBytes);
    // если ошибка, передаем управление обработчику ошибок
    if ( driverName2 == NULL) goto Error;
    // указываем номер порта
    driverName2->ConnectionIndex = ConnectionIndex;
    // получаем имя драйвера
    result = DeviceIoControl ( hHub,
      IOCTL_USB_GET_NODE_CONNECTION_DRIVERKEY_NAME, driverName2, ulBytes,
      driverName2, ulBytes, &ulBytes, NULL);
    // если ошибка, передаем управление обработчику ошибок
    if ( !result) goto Error;

    // конвертируем полученное имя
    driverName3 = _wStr_to_mStr ( driverName2->DriverKeyName);
    // освобождаем память
    GlobalFree ( driverName2);
    // возвращаем полученное имя драйвера
    return driverName3;
// обработчик ошибок
Error:
    if ( driverName2 != NULL)
    {
      GlobalFree ( driverName2);
      driverName2 = NULL;
    }
    return NULL; // завершаем функцию с ошибкой
}
```

```
char* USB :: _getRootHubName ( HANDLE HostController)
{
   bool result;
   unsigned long ulBytes = 0;
   char* HubName3 = NULL;
   USB_ROOT_HUB_NAME HubName;
   PUSB_ROOT_HUB_NAME HubName2 = NULL;
   // получаем требуемый размер буфера
   result = DeviceIoControl ( HostController,
      IOCTL_USB_GET_ROOT_HUB_NAME, 0, 0, &HubName, sizeof ( HubName),
      &ulBytes, NULL);
   // если ошибка, передаем управление обработчику ошибок
   if ( !result) goto Error;
   // получаем необходимый размер буфера для данных
   ulBytes = HubName.ActualLength;
   // выделяем память
   HubName2 = ( PUSB_ROOT_HUB_NAME) GlobalAlloc ( GPTR, ulBytes);
   // если ошибка, передаем управление обработчику ошибок
   if ( HubName2 == NULL) goto Error;
   // получаем имя хаба
   result = DeviceIoControl ( HostController,
     IOCTL_USB_GET_ROOT_HUB_NAME, NULL, 0, HubName2, ulBytes,
     &ulBytes, NULL);
   // если ошибка, передаем управление обработчику ошибок
   if ( !result) goto Error;
   // выполняем конвертирование строки
   HubName3 = _wStr_to_mStr ( HubName2->RootHubName);
   // освобождаем память
   GlobalFree ( HubName2);
   // возвращаем имя хаба
   return HubName3;
// обработчик ошибок
Error:
   if ( HubName2 != NULL)
   {
     GlobalFree ( HubName2);
     HubName2 = NULL;
   }
   return NULL; // выходим из функции с ошибкой
}
char*  USB :: _getKeyName ( HANDLE  hDevice)
{
   bool result;
   unsigned long ulBytes = 0;
```

```
   char* driverName3 = NULL;
   USB_HCD_DRIVERKEY_NAME  driverName;
   PUSB_HCD_DRIVERKEY_NAME driverName2 = NULL;
   // определяем размер буфера данных
   result = DeviceIoControl ( hDevice,
      IOCTL_GET_HCD_DRIVERKEY_NAME, &driverName, sizeof ( driverName),
      &driverName, sizeof ( driverName), &ulBytes, NULL);
   // если ошибка, передаем управление обработчику ошибок
   if ( !result) goto Error;
   // получаем необходимый размер буфера для данных
   ulBytes = driverName.ActualLength;
   // если полученный размер меньше требуемого
   // передаем управление обработчику ошибок
   if ( ulBytes <= sizeof ( driverName)) goto Error;
   // выделяем память
   driverName2 = ( PUSB_HCD_DRIVERKEY_NAME) GlobalAlloc ( GPTR,
                                                           ulBytes);
   // если ошибка, передаем управление обработчику ошибок
   if ( driverName2 == NULL) goto Error;
   // получаем имя драйвера
   result = DeviceIoControl ( hDevice, IOCTL_GET_HCD_DRIVERKEY_NAME,
     driverName2, ulBytes, driverName2, ulBytes, &ulBytes, NULL);
   // если ошибка, передаем управление обработчику ошибок
   if ( !result) goto Error;
   // выполняем конвертирование строки
   driverName3 = _wStr_to_mStr ( driverName2->DriverKeyName);
   // освобождаем память
   GlobalFree ( driverName2);
   // возвращаем имя драйвера
   return driverName3;
// обработчик ошибок
Error:
   if ( driverName2 != NULL)
   {
     GlobalFree(driverName2);
     driverName2 = NULL;
    }
   return NULL; // выходим из функции с ошибкой
}
char*  USB :: _wStr_to_mStr ( PWCHAR wStr)
{
   unsigned long ulBytes = 0;
   char* mStr = NULL;
```

```
   // определяем длину конвертируемой строки
   ulBytes = WideCharToMultiByte ( CP_ACP, 0, wStr, -1, NULL,  0, NULL,
                                   NULL);
   // если ноль, выходим из функции
   if ( ulBytes == 0) return NULL;
   // выделяем необходимый размер памяти
   mStr = ( PCHAR) GlobalAlloc ( GPTR, ulBytes);
   // если ошибка, выходим из функции
   if ( mStr == NULL) return NULL;
   // выполняем конвертирование
   ulBytes = WideCharToMultiByte ( CP_ACP, 0, wStr, -1, mStr, ulBytes,
                                   NULL, NULL);
   // если ошибка, освобождаем память и выходим из функции
   if ( ulBytes == 0)
   {
     GlobalFree ( mStr);
     return NULL;
   }
   return mStr;
}
// реализация открытых функций класса
void USB :: GetProductName ( unsigned int num, char* product)
{
   if ( !m_bIsGetParams) _getDevicesParams ();
   if ( num > m_uCountDevices) return;
   strcpy ( product, usb[num].ProductName);
}
void USB :: GetSerialNumber ( unsigned int num, char* serial)
{
   if ( !m_bIsGetParams) _getDevicesParams ();
   if ( num > m_uCountDevices) return;
   strcpy ( serial, usb[num].SerialNumber);
}
void USB :: GetVendorName ( unsigned int num, char* vendor)
{
   if ( !m_bIsGetParams) _getDevicesParams ();
   if ( num > m_uCountDevices) return;
   strcpy ( vendor, usb[num].VendorName);
}
unsigned short USB :: GetVersionNumber ( unsigned int num)
{
   if ( !m_bIsGetParams) _getDevicesParams ();
   if ( num > m_uCountDevices) return 0L;
```

```
    return usb[num].VersionNumber;
}
unsigned int USB :: GetCountDevices ()
{
    if ( !m_bIsGetParams) _getDevicesParams ();
    return m_uCountDevices;
}
unsigned long USB :: GetConnectedDevices ()
{
    if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
    return m_lCountDevicesConnected;
}
unsigned short USB :: GetProductID ( unsigned int num)
{
    if ( !m_bIsGetParams) _getDevicesParams ();
    if ( num > m_uCountDevices) return 0L;
    return usb[num].Product_ID;
}
unsigned short USB :: GetVendorID ( unsigned int num)
{
    if ( !m_bIsGetParams) _getDevicesParams ();
    if ( num > m_uCountDevices) return 0L;
    return usb[num].Vendor_ID;
}
void USB :: GetDriverName ( unsigned int num, char* driver)
{
    if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
    if ( num > m_lCountDevicesConnected) return;
    strcpy ( driver, drv[num].DriverName);
}
void USB :: GetRootHubName ( unsigned int num, char* root_hub)
{
    if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
    if ( num > m_lCountDevicesConnected) return;
    strcpy ( root_hub, drv[num].RootHubName);
}
void USB :: GetDeviceName_Ex ( unsigned int num, char* name)
{
    if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
    if ( num > m_lCountDevicesConnected) return;
    strcpy ( name, port[num].Device);
}
```

```
unsigned short USB :: GetVendor_Ex ( unsigned int num)
{
   if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
   if ( num > m_lCountDevicesConnected) return 0L;
   return port[num].Vendor;
}
unsigned short USB :: GetProduct_Ex ( unsigned int num)
{
   if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
   if ( num > m_lCountDevicesConnected) return 0L;
   return port[num].Product;
}
unsigned char USB :: GetClass_Ex ( unsigned int num)
{
   if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
   if ( num > m_lCountDevicesConnected) return 0L;
   return port[num].DeviceClass;
}
unsigned short USB :: GetAddress_Ex ( unsigned int num)
{
   if ( !m_bIsGetParamsEx) _getDevicesParamsEx ();
   if ( num > m_lCountDevicesConnected) return 0L;
   return port[num].Address;
}
bool USB :: OpenDevice ( unsigned int num)
{
   if ( !m_bIsGetParams) _getDevicesParams ();
   if ( m_hDevice != NULL) return false;
   HANDLE hDevice = NULL;
   // открываем устройство
   m_hDevice = CreateFile ( usb[num].DevicePath,
     GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ | FILE_SHARE_WRITE,
     ( LPSECURITY_ATTRIBUTES) NULL, OPEN_EXISTING, 0, NULL);
   if ( m_hDevice == INVALID_HANDLE_VALUE)
   {
     m_hDevice = NULL;
     return false; // не удалось открыть устройство
   }
   // устанавливаем номер текущего открытого устройства
   m_iCurrentOpenDevice = num;
   // выходим из функции
   return true;
 }
```

```
void USB :: CloseDevice ()
{
   if ( m_hDevice == NULL)
      return;
   else
   {
     CloseHandle ( m_hDevice);
     m_iCurrentOpenDevice = -1;
     m_bIsCaps = false;
     m_hDevice = NULL;
   }
}
bool USB :: GetDeviceCaps ()
{
   if ( ( m_hDevice == NULL) || ( m_iCurrentOpenDevice < 0))
       return false;
   PHIDP_PREPARSED_DATA Data = NULL;
   HIDP_CAPS Caps;
   ZeroMemory ( &Caps, sizeof ( HIDP_CAPS));
   ZeroMemory ( &usb_size, sizeof ( USBDATASIZE));
   if ( !HidD_GetPreparsedData ( m_hDevice, &Data))
        return false;
   if ( HidP_GetCaps ( Data, &Caps) != HIDP_STATUS_SUCCESS)
        return false;
   usb_size.FeatureByteLength = Caps.FeatureReportByteLength;
   usb_size.InputByteLength = Caps.InputReportByteLength;
   usb_size.OutputByteLength = Caps.OutputReportByteLength;
   m_bIsCaps = true;
   HidD_FreePreparsedData ( Data);
   return true;
}
bool USB :: GetFeature ( void* data)
{
   if ( ( m_hDevice == NULL) || ( m_iCurrentOpenDevice < 0) ||
         !m_bIsCaps) return false;
   if ( !data) return false;
   if( !HidD_GetFeature ( m_hDevice, &data, usb_size.FeatureByteLength))
        return false;
   return true;
}
bool USB :: SetFeature ( void* data, unsigned long data_len)
{
   if ( ( m_hDevice == NULL) || ( m_iCurrentOpenDevice < 0) ||
         !m_bIsCaps) return false;
```

```
   unsigned short len = 0;
   if ( data_len < usb_size.FeatureByteLength)
        len = usb_size.FeatureByteLength;
   else
       len = data_len;
   if( !HidD_SetFeature ( m_hDevice, data, len))
      return false;
   return true;
}
bool USB :: WriteData ( char* buffer, unsigned long buffer_len)
{
   if ( ( m_hDevice == NULL) || ( m_iCurrentOpenDevice < 0) ||
         !m_bIsCaps) return false;
   if ( !buffer || buffer_len > usb_size.OutputByteLength)
       return false;
   DWORD dwBytesWrite = 0;
   if ( !WriteFile ( m_hDevice, buffer, usb_size.OutputByteLength,
                     &dwBytesWrite, NULL))
   {
     return false;
   }
   return true;
}
bool USB :: ReadData ( char* buffer)
{
   if ( ( m_hDevice == NULL) || (  m_iCurrentOpenDevice < 0) ||
         !m_bIsCaps) return false;
   if ( !buffer || strlen ( buffer) < usb_size.InputByteLength)
       return false;
   DWORD dwBytesRead = 0;
   HANDLE hEvent = CreateEvent(NULL, true, true, NULL);
   if ( hEvent == NULL) return false;
   OVERLAPPED o;
   o.Offset = 0;
   o.OffsetHigh = 0;
   o.hEvent = hEvent;
   if( !ReadFile ( m_hDevice, buffer, usb_size.InputByteLength,
                   &dwBytesRead, &o))
   {
      DWORD dwResult = WaitForSingleObject ( hEvent, 3000);
        switch ( dwResult)
        {
         case WAIT_OBJECT_0:
               break;
```

```
        case WAIT_TIMEOUT:
              CancelIo ( m_hDevice);
              ResetEvent( hEvent);
              CloseHandle ( hEvent);
              hEvent = NULL;
              return false;
        }
      }
    ResetEvent( hEvent);
    CloseHandle ( hEvent);
    hEvent = NULL;
    if ( !GetOverlappedResult ( m_hDevice, &o, &dwBytesRead, false))
       return false;
   return true;
}
```

Реализация поддержки устройств USB в представленном классе осуществляется двумя способами: посредством набора функций менеджера установки устройств и с помощью универсальной функции ввода-вывода `DeviceIoControl`. Рассмотрим каждый из них более подробно.

В первом случае вызывается функция `_getDevicesParams`. Она находит все устройства определенного класса и выделяет их основные параметры (идентификаторы производителя и изделия) для дальнейшего доступа. Чтобы явно указать класс USB-устройств, мы применили специальную функцию `HidD_GetHidGuid`, которая предназначена для определения уникального глобального идентификатора (`GUID`), поддерживающего все устройства с интерфейсом USB. Далее мы передали `GUID` в функцию `SetupDiGetClassDevs` для получения информации об устройствах указанного класса (в нашем случае USB). На следующем этапе, периодически вызывая функцию `SetupDiEnumDeviceInterfaces`, определяем параметры всех имеющихся в системе USB-устройств. Для этого дополнительно применяем функцию `SetupDiGetDeviceInterfaceDetail`. Главная ее задача состоит в возвращении пути к найденному устройству. Используя путь, открываем устройство (с помощью `CreateFile`) и получаем основные параметры посредством функции `HidD_GetAttributes`. Дополнительно обрабатываем информацию о производителе и серийном номере устройства (`HidD_GetManufacturerString`, `HidD_GetProductString` и `HidD_GetSerialNumberString`). И, в завершение, закрываем текущее устройство и переходим к поиску следующего. Когда все устройства найдены, мы можем приступать к непосредственной работе с ними. Сначала следует проанализировать возможности устройства, особенно если вы планируете выполнять обмен данными. Эта задача решается в функции `GetDeviceCaps`. Вызываем `HidD_GetPreparsedData` (предварительная подготовка данных об устройстве) и в случае успешного завершения опре-

деляем непосредственно поддерживаемые устройством возможности (HidP_GetCaps). В данном случае нас интересуют только три параметра: размер данных для управления свойствами устройства (FeatureReportByteLength), минимальные размеры в байтах входного (InputReportByteLength) и выходного (OutputReportByteLength) буферов для обмена данными. Для получения и установки свойств используются соответственно функции HidD_GetFeature и HidD_SetFeature. Обмен данными с устройствами USB осуществляется с помощью стандартных файловых функций Win32: ReadFile и WriteFile. Вот, в принципе, и все базовые сведения. Естественно, для организации передачи или приема данных необходимо иметь представление о применяемом в USB протоколе и правилах форматирования последовательности байтов. Получить эту, а также другую полезную информацию о шине USB можно, обратившись по адресу **www.usb.org**.

Второй способ основан на использовании функции DeviceIoControl. Принцип работы этой функции базируется на прямом взаимодействии программы и устройства (драйвера) посредством специальных управляющих кодов. Например, для сброса порта USB служит код IOCTL_INTERNAL_USB_RESET_PORT. Полный список всех поддерживаемых кодов можно посмотреть в файле usbioctl.h, входящем в состав пакета DDK.

В любом случае показанный выше пример класса не охватывает и половины всех возможностей интерфейса USB, но дает общее представление о методах разработки программ для данного типа устройств.

* * *

На этом позвольте мне завершить описание вопросов программирования в операционных системах Windows. Надеюсь, что представленный вашему вниманию материал поможет решить многие повседневные проблемы при разработке современного программного обеспечения. Основываясь на собственном опыте, я постарался рассказать о наиболее интересных и важных темах, мало освещенных в других изданиях. Хочется верить, что эта книга будет востребована как начинающими, так и профессиональными программистами. Желаю всем читателям никогда не останавливаться на достигнутом, поскольку только так можно добиться настоящего успеха.

ПРИЛОЖЕНИЕ

Описание компакт-диска

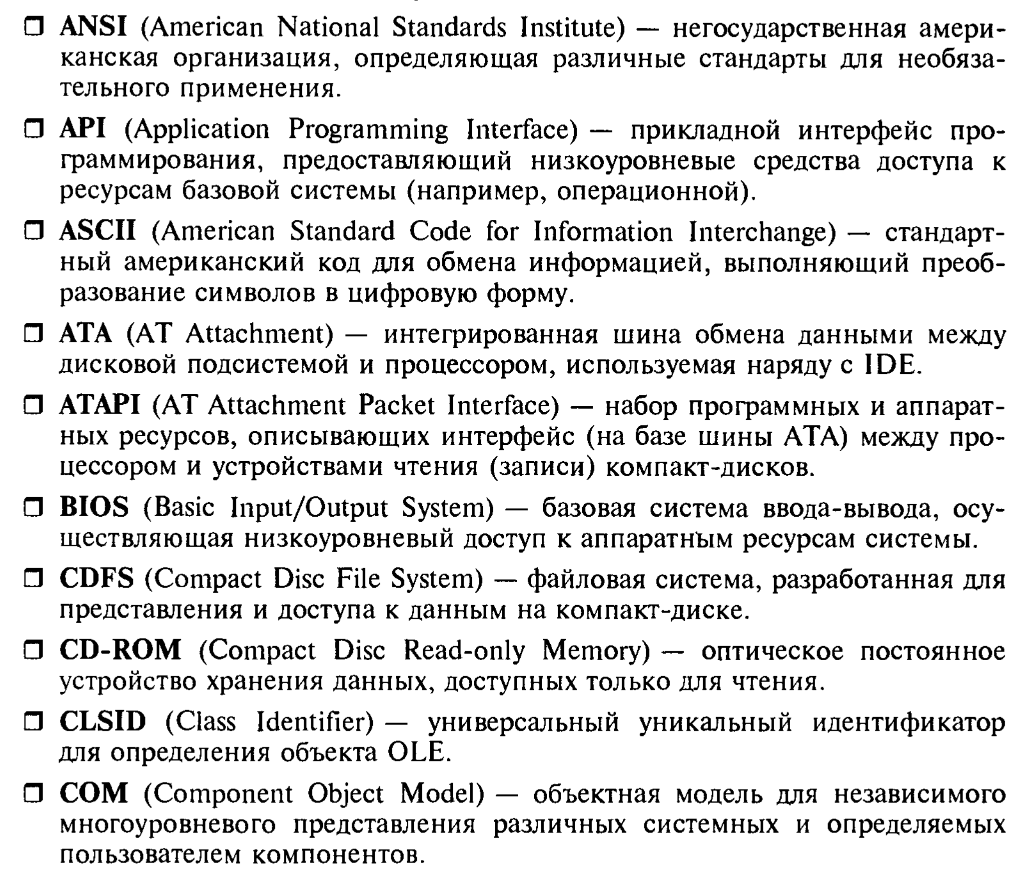
Прилагаемый к книге диск содержит исходные коды всех примеров и системные драйверы для работы с аппаратными портами ввода-вывода. Примеры расположены в каталоге **Examples**, а драйверы — в **Drivers**. Нумерация примеров совпадает с соответствующими номерами листингов. Выбор драйвера зависит от установленной операционной системы: для Windows 9*x*/ME предназначен Io32port.vxd, а для Windows NT/2000/XP/SR3 — IOtrserv.sys. Чтобы использовать драйвер, достаточно скопировать его в папку с проектом, в его свойствах снять атрибут "только для чтения" и добавить в свой проект код класса `CIO32` или `CIO32NT`, описанный в *гл. 1*.

Просмотр диска осуществляется только в ручном режиме:

1. Вставьте диск в устройство чтения.
2. Откройте папку **Мой компьютер** (My Computer).
3. Выберите устройство чтения компакт-дисков и откройте его.

Хочу обратить ваше внимание, что предлагаемые системные драйверы предназначены только для тестирования примеров из данной книги и не должны быть использованы как-либо иначе.

Глоссарий



- **AGP** (Accelerated Graphics Port) — специально выделенный высокоскоростной порт для наилучшей передачи графических трехмерных данных от видеоадаптера в системную память.
- **ANSI** (American National Standards Institute) — негосударственная американская организация, определяющая различные стандарты для необязательного применения.
- **API** (Application Programming Interface) — прикладной интерфейс программирования, предоставляющий низкоуровневые средства доступа к ресурсам базовой системы (например, операционной).
- **ASCII** (American Standard Code for Information Interchange) — стандартный американский код для обмена информацией, выполняющий преобразование символов в цифровую форму.
- **ATA** (AT Attachment) — интегрированная шина обмена данными между дисковой подсистемой и процессором, используемая наряду с IDE.
- **ATAPI** (AT Attachment Packet Interface) — набор программных и аппаратных ресурсов, описывающих интерфейс (на базе шины ATA) между процессором и устройствами чтения (записи) компакт-дисков.
- **BIOS** (Basic Input/Output System) — базовая система ввода-вывода, осуществляющая низкоуровневый доступ к аппаратным ресурсам системы.
- **CDFS** (Compact Disc File System) — файловая система, разработанная для представления и доступа к данным на компакт-диске.
- **CD-ROM** (Compact Disc Read-only Memory) — оптическое постоянное устройство хранения данных, доступных только для чтения.
- **CLSID** (Class Identifier) — универсальный уникальный идентификатор для определения объекта OLE.
- **COM** (Component Object Model) — объектная модель для независимого многоуровневого представления различных системных и определяемых пользователем компонентов.

- **DDC** (Display Data Channel) — канал обмена данными между дисплеем и видеоадаптером.
- **DDK** (Device Driver Kit) — набор программных компонентов для разработки драйверов. Для разных версий Windows имеются собственные пакеты разработчика драйверов.
- **DIB** (Device-independent Bitmap) — формат графического представления растровых изображений в виде, не зависимом от устройства отображения. Другими словами, любая графика в этом формате будет одинаково выглядеть на различных системах.
- **DLL** (Dynamic Link Library) — двоичный тип файла, динамически загружаемый по мере необходимости в область памяти вызывающей программы. Он используется в первую очередь для разделения ресурсов и программных интерфейсов, тем самым уменьшая размер исполняемого файла.
- **DMA** (Direct Memory Access) — прямой доступ к памяти, позволяющий устройству передавать или получать данные без участия процессора, что значительно экономит время и системные ресурсы.
- **DPMS** (Display Power Management Signaling) — стандарт управления питанием дисплея.
- **DSP** (Digital Signal Processor) — цифровой сигнальный процессор, предназначенный для цифровой обработки при высокоскоростной передаче данных. Наиболее часто применяется в устройствах связи и звуковых платах.
- **ECP** (Extended Capabilities Port) — асинхронный 8-разрядный канал для параллельной передачи данных от компьютера на устройство и обратно.
- **FAT** (File Allocation Table) — таблица размещения файловых объектов на носителе, позволяющая упорядочить хранение и доступ к данным.
- **FDC** (Floppy disk Drive Controller) — контроллер накопителя на гибких дисках.
- **FTP** (File Transfer Protocol) — протокол передачи файлов из сети на компьютер и обратно посредством TCP/IP.
- **GUID** (Globally Unique Identifier) — уникальный глобальный идентификатор, представляюший собой 16-байтное значение, гарантирующее корректный доступ к указанному объекту в операционной системе.
- **HCD** (Host Controller Driver) — устройство ведушего контроллера. В контексте шины USB оно предназначено для задающего устройства управления чипсетом периферийного оборудования.
- **HCI** (Host Controller Interface) — интерфейс ведущего контроллера, используемый на шине USB.

- **HDC** (Hard Disk I/O Controller) — контроллер ввода-вывода жесткого диска.
- **HID** (Human Interface Device) — интерфейс устройства, используемого на шине USB.
- **HTTP** (Hypertext Transfer Protocol) — протокол Интернета, используемый браузерами и серверами для обмена информацией.
- **I2C** (Inter-Integrated Circuit Bus) — последовательная шина передачи данных по двум проводам (синхронизация и данные).
- **ICMP** (Internet Control Message Protocol) — расширение протокола IP для передачи данных, поддерживающих генерацию сообщений об ошибках, тестовых пакетов и информационных сообщений.
- **IDE** (Integrated Device Electronics) — дисковое устройство хранения данных с интегрированным контроллером.
- **IID** (Interface Identifier) — уникальный глобальный идентификатор, связанный с определенным интерфейсом.
- **IP** (Internet Protocol) — базовый протокол Интернета, позволяющий передавать пакеты данных от одного хост-компьютера к другому.
- **LCID** (Locale Identifier) — идентификатор (32-разрядный) языка для правильного отображения информации.
- **MIDI** (Musical Instrument Digital Interface) — цифровой интерфейс описания музыкальных инструментов, позволяющий с помощью компьютера управлять различными музыкальными устройствами.
- **NMI** (Nonmaskable Interrupt) — немаскируемое прерывание. Данный тип прерывания является приоритетным и не может быть аннулирован другим запросом на обслуживание.
- **OLE** (Object Linking and Embedding) — технология, позволяющая встраивать внешние (принадлежащие другим приложениям) объекты в собственную программу.
- **PCI** (Peripheral Component Interconnect) — межкомпонентная 32- или 64-разрядная шина, объединяющая различные устройства.
- **PIC** (Programmable Interrupt Controller) — программируемый контроллер прерываний.
- **POST** (Power-On Self-Test) — самотестирование компьютерной системы после включения питания, инициируемое BIOS.
- **SMTP** (Simple Mail Transfer Protocol) — стандартный протокол Интернета для передачи электронной почты.
- **SVGA** (Super VGA) — стандарт дисплея с высоким разрешением (не менее 1024 × 768) и качеством изображения.

- **TCP/IP** (Transmission Control Protocol/Internet Protocol) — набор стандартных транспортных протоколов для Интернета, состоящий из следующих компонентов: TCP, IP, UDP и ICMP.
- **UDP** (User Datagram Protocol) — транспортный протокол без установки соединения, формирующий интерфейс пользователя для протокола IP.
- **URL** (Uniform Resource Identifier) — идентификатор ресурса, используемый в глобальной сети.
- **USB** (Universal Serial Bus) — универсальная последовательная шина, поддерживающая двунаправленный обмен данными и динамическое подключение (при включенном питании компьютера) устройств.
- **VGA** (Video Graphics Array) — стандарт дисплея, поддерживающего разрешение цветного экрана не менее 640 × 480 пикселов.
- **Win32 API** — прикладной 32-разрядный интерфейс программирования для всех операционных систем Windows.

Список литературы

1. Гук М. Аппаратные средства IBM PC. Энциклопедия. — СПб.: Питер Ком, 1999.
2. Зубков С. В. Assembler для DOS, Windows и UNIX. — 2-е изд., испр. и доп. — М.: ДМК, 2000.
3. Несвижский В. Программирование устройств SCSI и IDE. — СПб.: БХВ-Петербург, 2003.
4. Страуструп Б. Язык программирования C++, 3-е изд.: Пер. с англ. — СПб.; М.: "Невский Диалект" — "Издательство БИНОМ", 1999.
5. Трельсен Э. Модель COM и библиотека ATL 3.0. — СПб.: БХВ-Петербург, 2001.
6. Шилдт Г. Справочник программиста по C / C++. Пер. с англ.: Учеб. пособие. — М.: Издательский дом "Вильямс", 2000.

Предметный указатель

Продолжение рубрики см. на с. 874

М

П

Продолжение рубрики см. на с. 876

Продолжение рубрики см. на с. 878